Experimental Syntax Using IBEX / PCIBEX

SICOL 2022

Nayoun Kim and June Choe

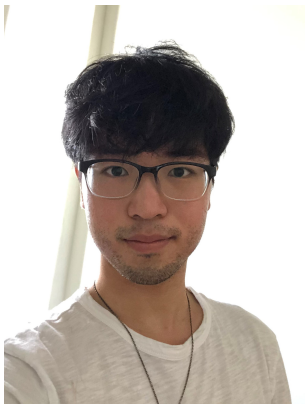
August 12, 2022

About us

Nayoun Kim nayoun@skku.edu

Assistant Professor, Sungkyunkwan University
Department of English Language and Literature

Syntax, Experimental Syntax, Psycholinguistics



June Choe yjchoe@sas.upenn.edu

Ph.D. Student, University of Pennsylvania
Department of Linguistics

Psycholinguistics, Acquisition, Computational Linguistics

Presentation Outline

Introduction to IBEX and PCIbex

- Basic ideas (IBEX code + PCIbex farm)
- Navigating the platform
- File structure

Scripting an experiment

- Overview of critical components
- Code walkthrough

Analysis of the results

- Understanding the output format
- Importing into Excel and R

1. Introduction to IBEX and PCIbex

What is IBEX and PCIbex?

IBEX

- (**I**)nternet-(**B**)ased (**EX**)periments developed by Alex Drummond.
- Refers to both the **framework for scripting experiments** and the **platform for hosting experiments**.
- IBEX the *platform* "IBEXfarm" has been discontinued in late 2021, but the IBEX *framework* survives and is supported by extensions and spinoffs.

PCIbex

- PennController for IBEX developed by Jeremy Zehr and Florian Schwarz at the University of Pennsylvania.
- A platform for hosting experiments that **supports and extends IBEX**
- Also introduces its own, alternative framework for scripting experiments.

Language

Framework

IBEX

PCIbex

Platform (hosting)

IbexFarm

PCIbex farm

Endpoint

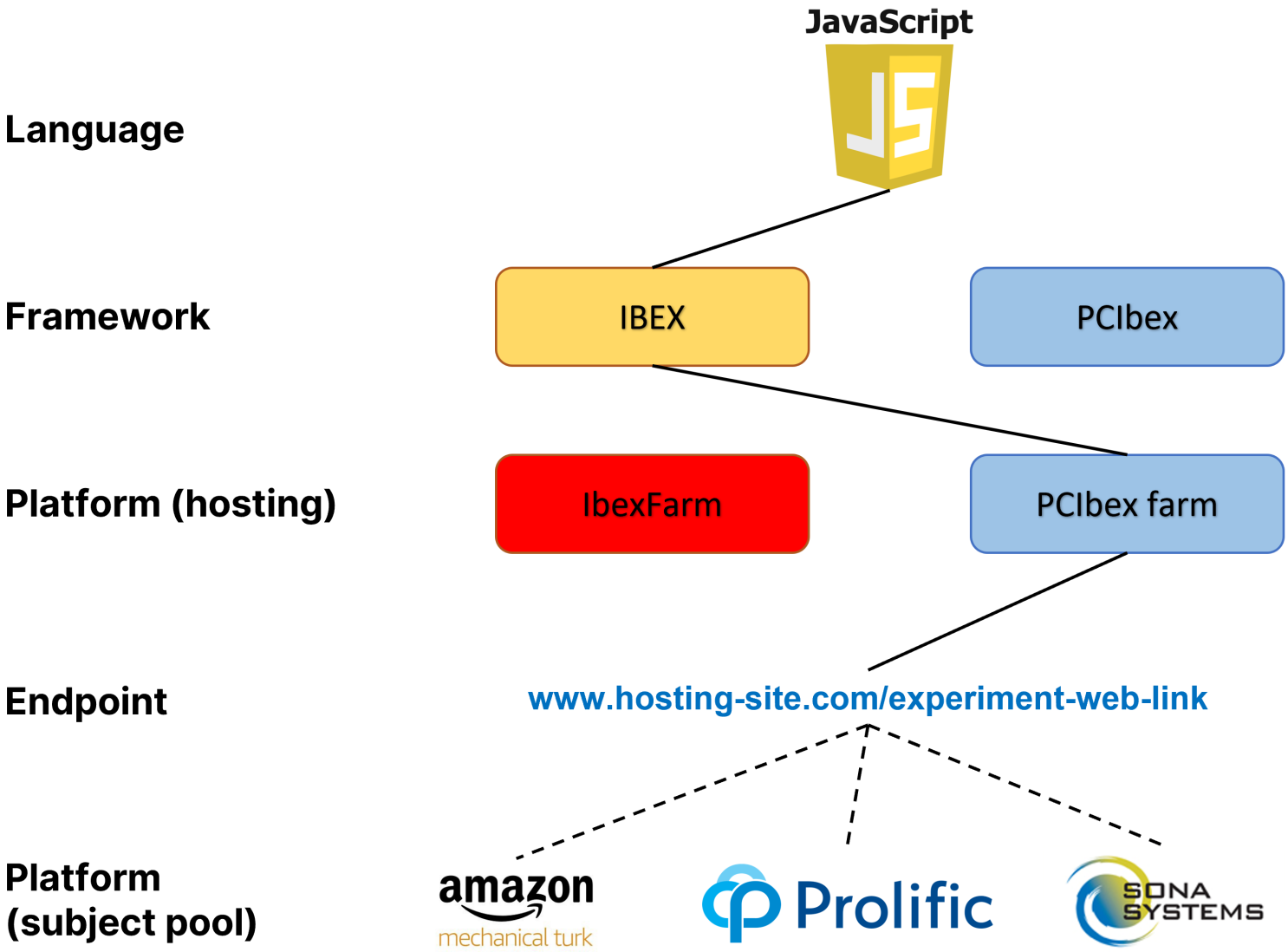
www.hosting-site.com/experiment-web-link

Platform (subject pool)

amazon mechanical turk

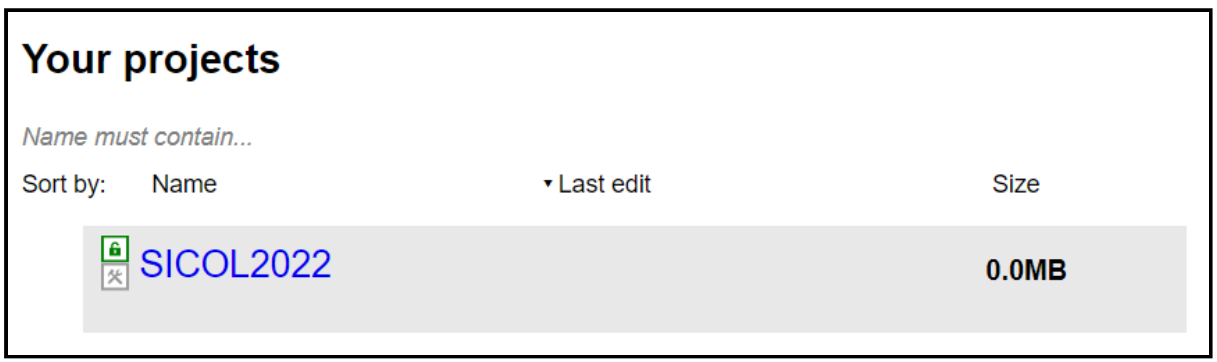
Prolific

SONA SYSTEMS

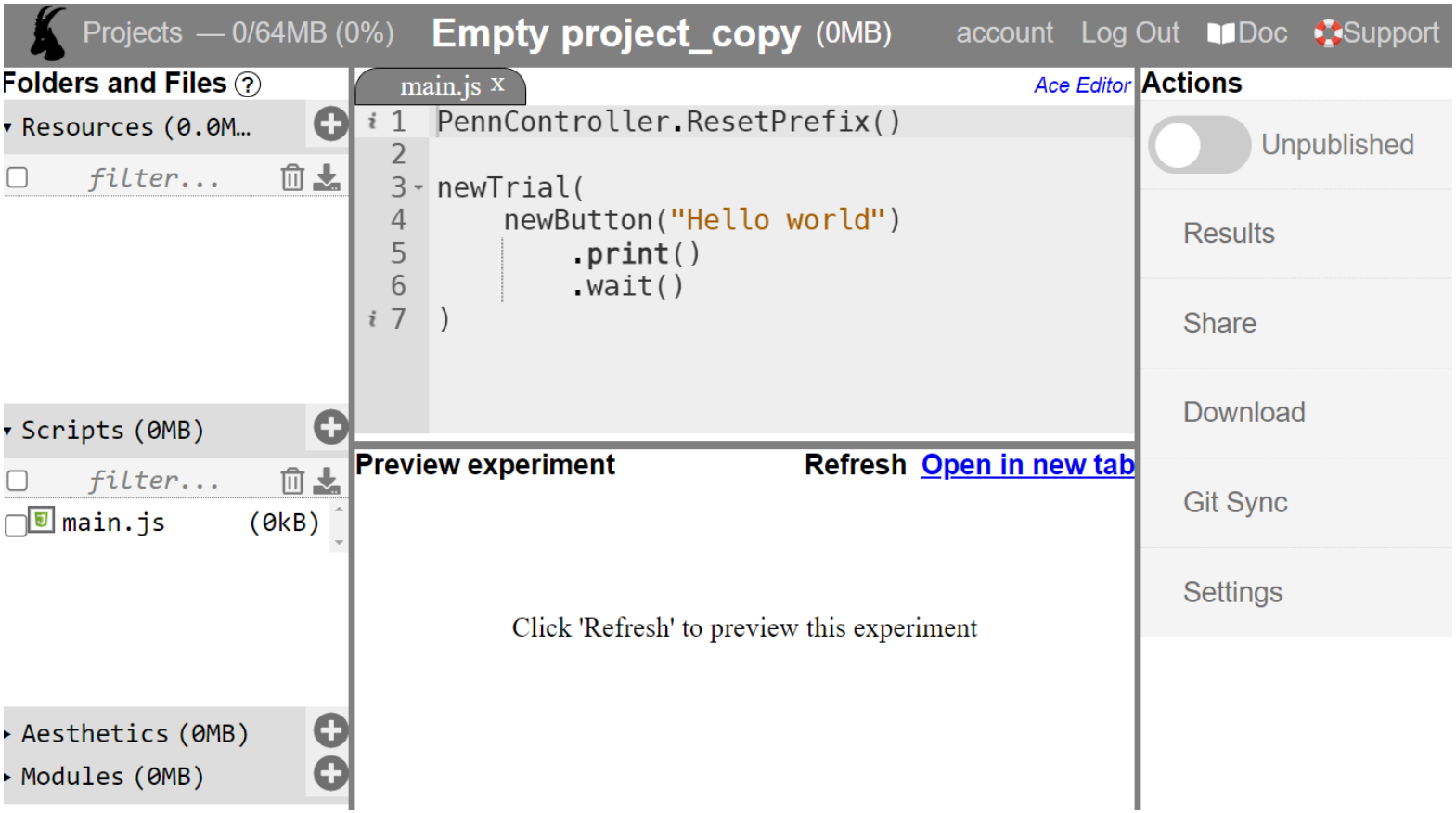

Language
JavaScript
Framework
IBEX
PCIbex
Platform (hosting)
IbexFarm
PCIbex farm
Endpoint
www.hosting-site.com/experiment-web-link
Platform
(subject pool)
amazon
mechanical turk
Prolific
SONA
SYSTEMS

Navigating the PCIbex platform

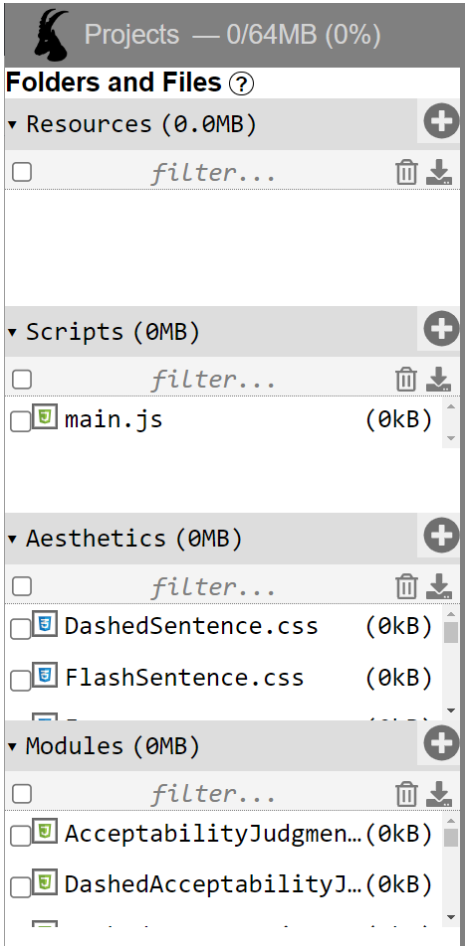
- Go to the PCIbex farm https://farm.pcibex.net
- **Sign up** with an email (or **Log in**)
- Once logged in, click **Empty project** under **Start a new project**
- Change the name of the project at the top to something unique (e.g., "SICOL2022")
- Your new project will appear on the main page



PCIbex Dashboard Interface



Experiment file structure



Resources

- Files for experiment materials go here

Scripts

- Experiment scripts go here

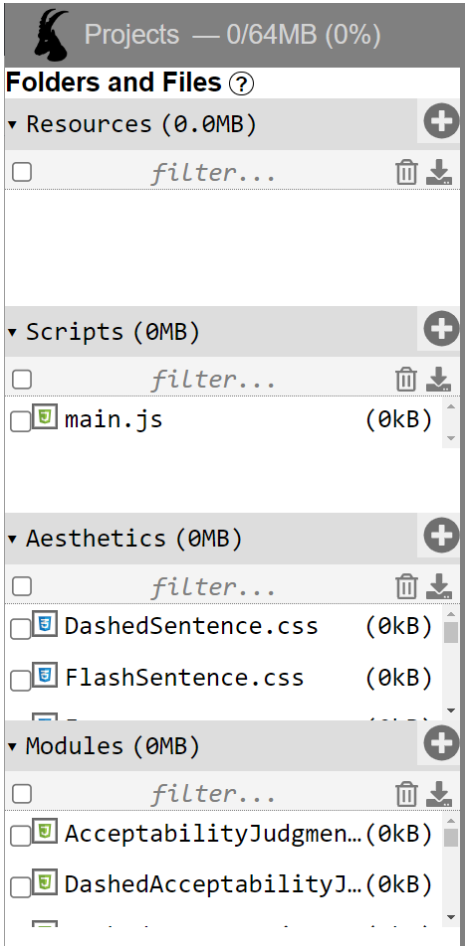
Aesthetics

- Style sheets go here

Modules

- Controllers (a.k.a. "Elements") go here

Experiment file structure



Resources

- Files for experiment materials go here

Scripts

- Experiment scripts go here

Aesthetics

- Style sheets go here

Modules

- Controllers (a.k.a. "Elements") go here

2. Scripting an experiment

The script

When scripting with IBEX, we often only need **one file**: *main.js*

At creation, the default file looks like this:

```
PennController.ResetPrefix()

newTrial(
    newButton("Hello world")
        .print()
        .wait()
)
```

This is a snippet of code from the **PCIbex scripting framework**, which is not the focus of the workshop today (we will briefly revisit PCIbex at the end).

Writing your own script

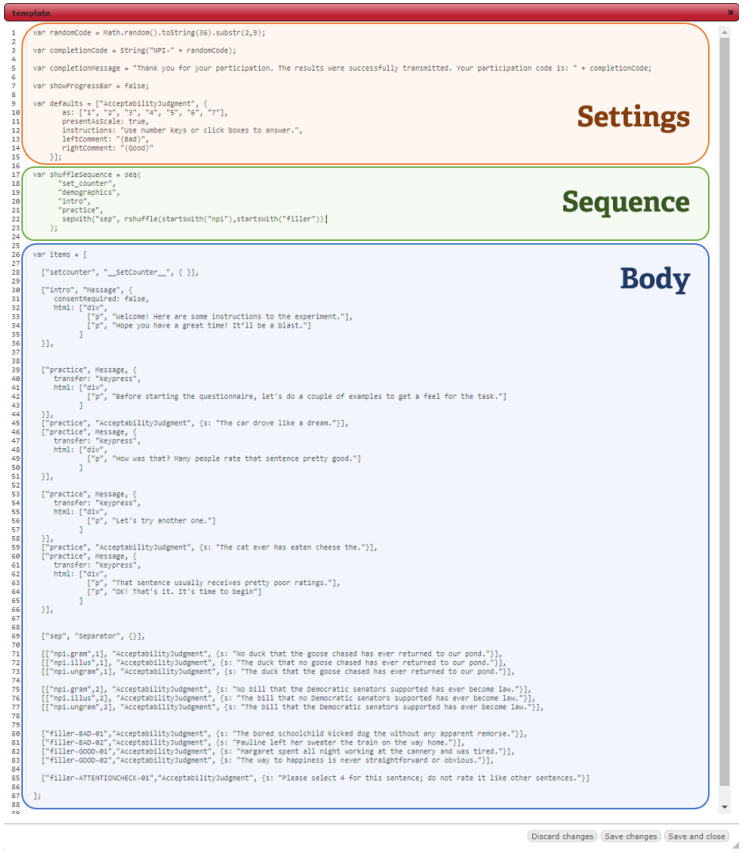
To demonstrate the **IBEX scripting framework**, we use our own **template file**.

The following IBEX code *just works* thanks to PCIbex's backward compatibility.

```
// Defaults and other settings //

//// A message to show to participants at completion
var completionMessage = "Thank you for your participation!"

//// 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)"
    }]
```

This script creates an acceptability judgment experiment.

A simple layout



Parts of the `main.js` Script

Settings:

- Sets various options for the experiment

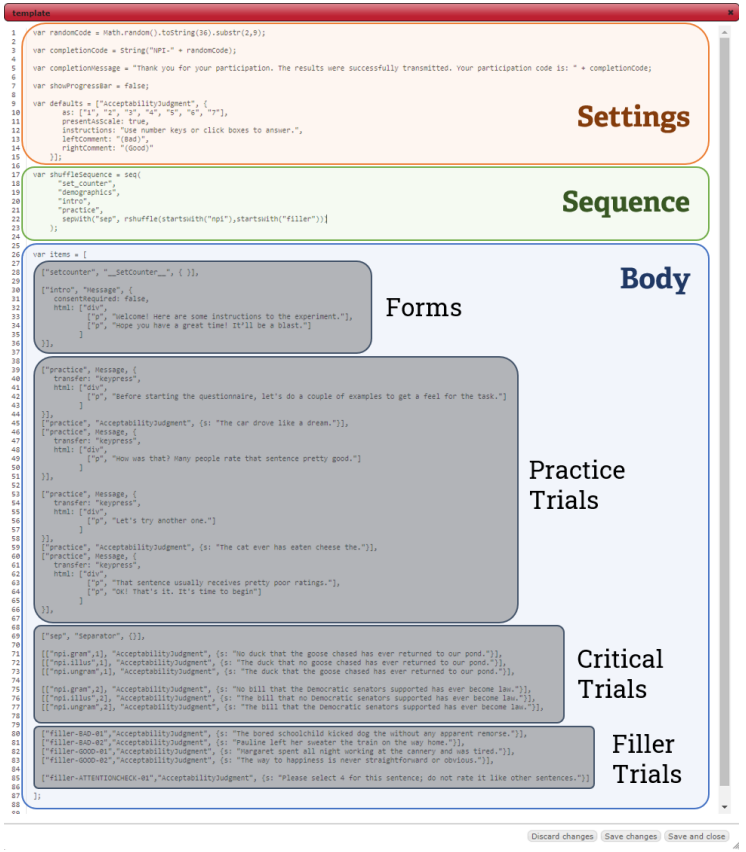
Sequence:

- Specifies the ordering of the different parts of the experiment

Body:

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

A simple layout



Body

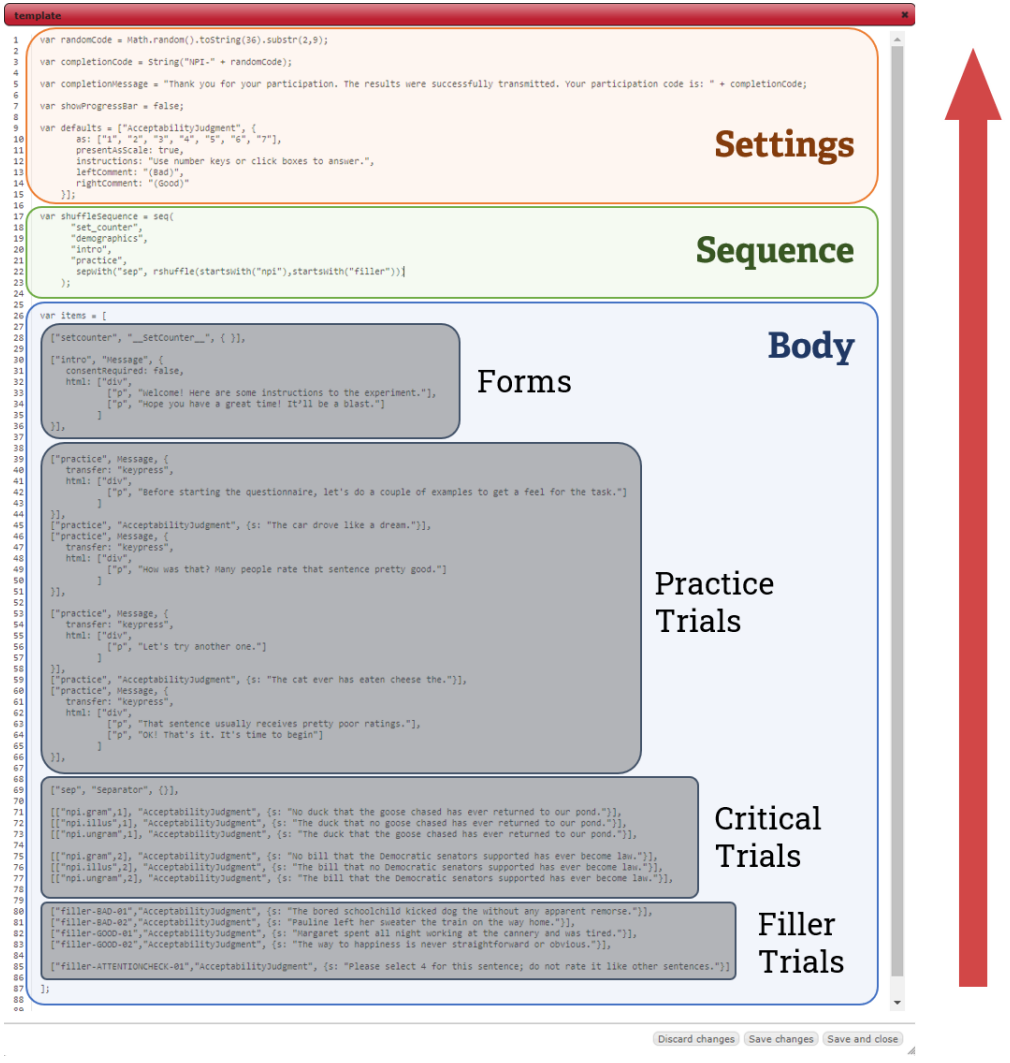
Forms

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

Trials

- Stimuli for the experiment
  - **practice** trials
  - **critical** trials
  - **filler** trials

Walkthrough of the components



Our first experiment

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 (reanalysis) more difficult, compared to verbs that are frequently **intransitive**.

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

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

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

We also want a *within-participant* design!

Trial Syntax

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

[["*Condition name*", *Item Set #*], "*Task Type*", {s: "*Sentence*"}]

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

```
[
    ["Condition name", Item Set # ],
    "Task Type",
    {s: "Sentence"}
]
```

```
[ ["Condition name", Item Set # ],
    "Task Type",
    {s: "Sentence"} ]
```

We have one big list (bracket) which contains **three elements**:

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

Translating the design to code

Example 1:

In 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:

In 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:

In 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:

In 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:

In 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:

In 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
```

Another example set

Item set #2:

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

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

```
[["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."}]
```

Putting together the stimuli

Wrap in `var items = [ ... ]` and separate each item 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! (scripting outside of PCIbex using an editor is also an option)

Practice and Fillers

Practice Trials:

- Presented at the beginning, accompanied by instructions and feedback

Filler Trials:

- Mixed in with critical trials, to distract or exclude participants

We want both practice and filler trials to be invariant across conditions

To do that, we don't specify *Item Set #* for give practice and filler trials, as that gets used for counterbalancing (more later):

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."}]
```

Practice and Fillers

Practice Trials:

- Presented at the beginning, accompanied by instructions and feedback

Filler Trials:

- Mixed in with critical trials, to distract or exclude participants

We want both practice and filler trials to be invariant across conditions

To do that, we don't specify *Item Set #* for give practice and filler trials, as that gets used for counterbalancing (more later):

Practice | Filler-good | Filler-bad | Filler-catch

Good Filler #1: "When Harry fell, the audience was shocked."

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

Practice and Fillers

Practice Trials:

- Presented at the beginning, accompanied by instructions and feedback

Filler Trials:

- Mixed in with critical trials, to distract or exclude participants

We want both practice and filler trials to be invariant across conditions

To do that, we don't specify *Item Set #* for give practice and filler trials, as that gets used for counterbalancing (more later):

Practice | Filler-good | Filler-bad | Filler-catch

Bad Filler #1: "When Tyler sneezed the driver, he passed a tissue."

```
["filler-bad-01",
"AcceptabilityJudgment",
{s: "When Tyler sneezed the driver, he passed a tissue."}]
```

Practice and Fillers

Practice Trials:

- Presented at the beginning, accompanied by instructions and feedback

Filler Trials:

- Mixed in with critical trials, to distract or exclude participants

We want both practice and filler trials to be invariant across conditions

To do that, we don't specify *Item Set #* for give practice and filler trials, as that gets used for counterbalancing (more later):

Practice | Filler-good | Filler-bad | Filler-catch

Catch Filler #1: "Please select 4 for this sentence."

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

Putting together the Body

```
var items = [

  //// 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 the Sequence

```
var items = [

  //// 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-
```

Pull out the *names* of each trial:

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

Defining the Sequence

```
var items = [

  //// 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-
```

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 the Sequence

```
var items = [

  //// 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-
```

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 the Sequence

```
var items = [

  //// 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-
```

Problems

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

1. A lot of typing ("-02", "-03", ...)

Defining the Sequence

```
var items = [

  //// 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-
```

Problems

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

1. A lot of typing ("-02", "-03", ...)
2. Presentation order of some trials should be random

Defining the Sequence

```
var items = [

  //// 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-
```

Problems

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

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

1. 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.

2. 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")
   )
)
```

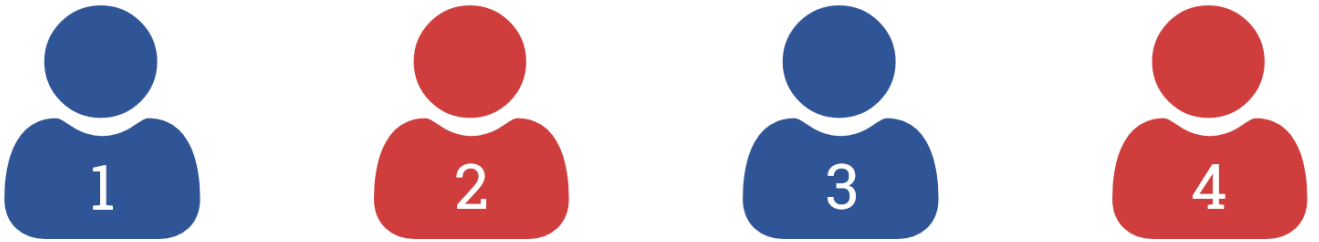
By wrapping both the critical trials (*gp...*) and the filler trials (*filler...*) in `rshuffle()`, they are mixed together and presented in random order.

We can also write this out more compactly:

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

3. Sequence: counterbalancing

The **set number** in our critical trials are automatically used for **counterbalancing**:



To alternate items between participants, we use a special **counter** "trial":

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

3. Sequence: counterbalancing

The counter must be defined in `items` and added to `shuffleSequence`:

```
var items = [
  ...
  ["setcounter", "__SetCounter__", { }], // The counter is defined
  ...
]

var shuffleSequence = seq(
  "setcounter", // The counter is incremented at the start
  ...
)
```

Sometimes you want to increment counter 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."}],
```

Just need one more step: **Settings**

Settings

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

For example, using the `defaults` variable, we can specify parameters for the **design** of the tasks that we use in the experiment, like **Acceptability Judgment**.

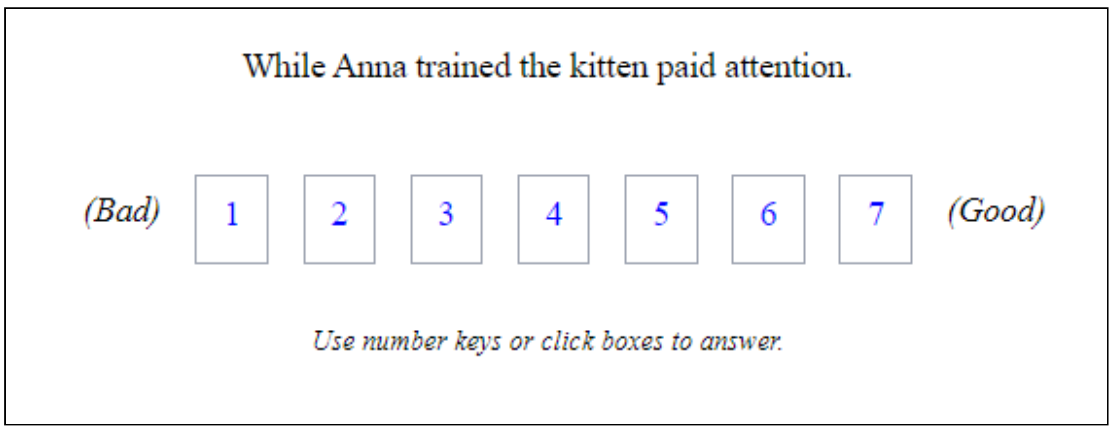
```
// Defaults and other settings //

//// Override/set 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.

The `defaults` variable

Some options for **Acceptability Judgment**:

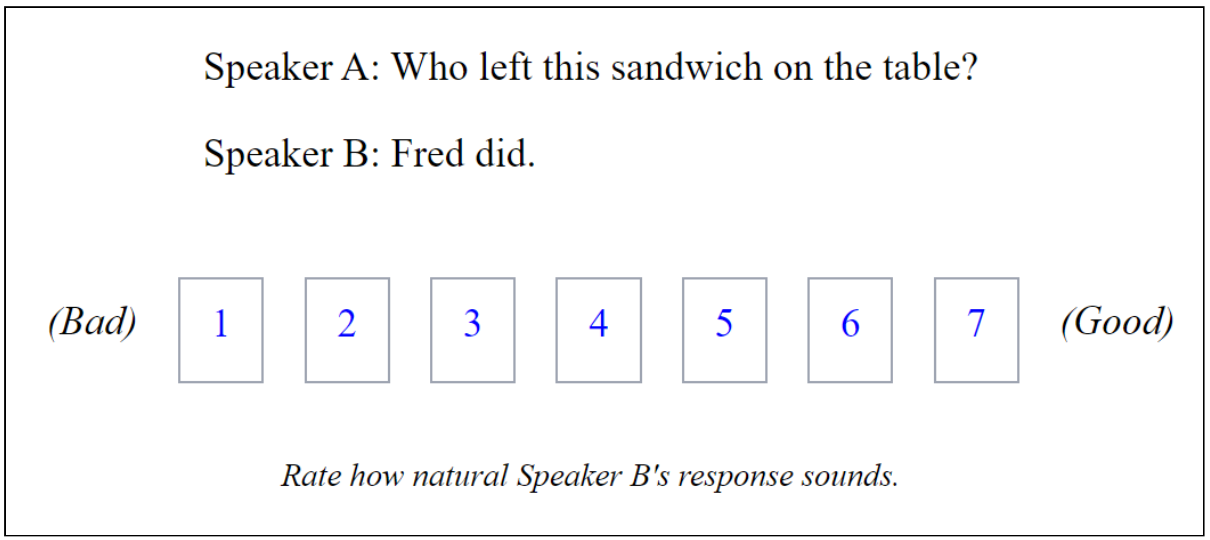
```
var defaults = [
    "AcceptabilityJudgment", {
       as: ["1", "2", "3", "4", "5", "6", "7"],
       presentAsScale: true,
       instructions: "Use number keys or click boxes to answer.",
       leftComment: "(Bad)",
       rightComment: "(Good)"
    }]
```



Another `defaults` example

Suppose you'd like to show participants *multiple sentences* but have them only rate the acceptability of **one** of the sentences.

Output | Code



Another `defaults` example

Suppose you'd like to show participants *multiple sentences* but have them only rate the acceptability of **one** of the sentences.

Output | Code

```
var defaults = [
    "AcceptabilityJudgment", {
        ...
        instructions: "Rate how natural Speaker B's response sounds."
    }]
```

```
[["ConditionA", 1],
 "AcceptabilityJudgment",
 {s: ["div",
        ["p", "Speaker A: Who left this sandwich on the table?"]
        ["p", "Speaker B: Fred did."]
      ]}]
```

We can print lines of text using **html** tags (more later)

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
var completionMessage = "Thank you for your participation!"
```

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 IBEX documentation.

A minimal experiment

We now have a minimally working experiment!

```
// Defaults and other settings //

//// A message to show to participants at completion
var completionMessage = "Thank you for your participation!"

//// 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",
  startsWith("practice"),
  rshuffle(startsWith("gp"), startsWith("filler"))
)
```

Interim Summary #1

What we've covered:

- We have a templatic syntax for creating stimuli
- We store all the materials for our experiment inside `items`
- We specify the order of presentation inside `shuffleSequence`
- We can tweak various aspects of the experiment, like `defaults` for the *task design*.

A few more things we want to know:

- How can we show *messages and text* to participants?
  - Introduction page, consent form, directions, etc.
- How can we extend this workflow for *other experimental designs*?
  - Self-paced reading, comprehension tasks, etc.

The "Message" controller

The **"Message"** controller shows text on a new page.

It is a list of 3 elements, similar to the "AcceptabilityJudgement" items:

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

- ***Condition name*** is used to reference the trial in sequencing
- **"Message"** is a type of task that just shows text on a screen
- Inside of the curly braces **{}** we can add text in the **html** parameter:

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

Notes on **html**:

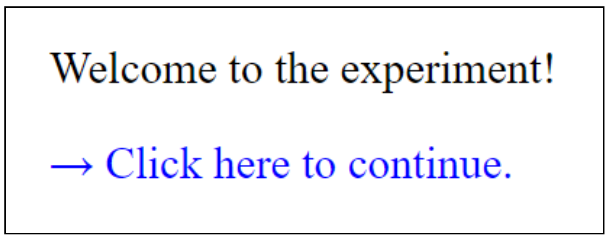
- The code `["p", "<your text here>"]` prints a paragraph of text.
- The "p" is an **HTML tag** and there are many others (most common: "div", "strong", "em"). They don't usually get too complicated.

Message examples

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

A message composed of a single paragraph:

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

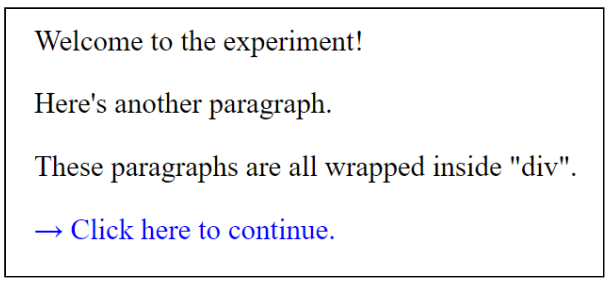


Message examples

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

A message composed of multiple paragraphs must be combined inside a "div":

```
["intro", "Message", {html:
    ["div",
     ["p", "Welcome to the experiment!"],
     ["p", "Here's another paragraph."],
     ["p", "These paragraphs are all wrapped inside \"div\"."]
    ]
}]
```



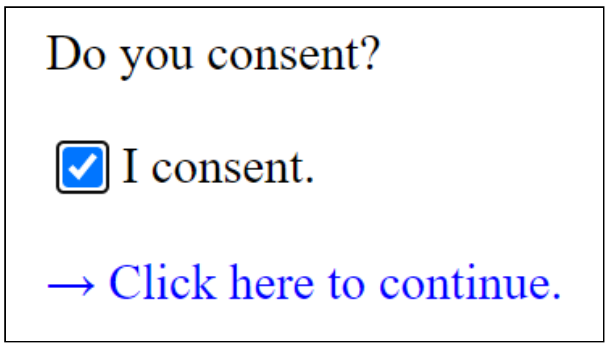
Note: You can **escape** special characters like quotes " with a backslash \

Message examples

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

Ask for consent with a set of `consent...` parameters

```
["consent", "Message", {
  html: ["p", "Do you consent?"],
   consentRequired: true,
   consentMessage: "I consent."
}]
```



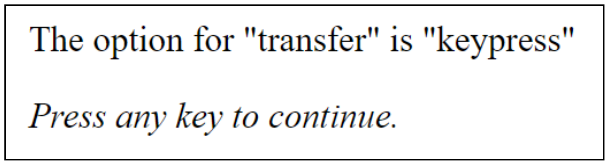
Message examples

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

Use the `transfer` argument to specify how the participant can move on.

Setting `transfer` to `"keypress"` removes the default "**Click here to continue.**" message and allows participants to proceed with the press of any key.

```
["move_on", "Message", {
  html: ["div",
          ["p", "The option for \"transfer\" is \"keypress\""],
          ["em", "Press any key to continue."]
        ],
  transfer: "keypress"
}]
```

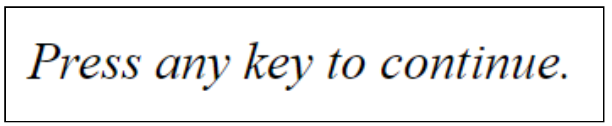


Message examples

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

You might want to insert a page that separates the trials:

```
["sep", "Message", {
  html: ["em", "Press any key to continue."],
  transfer: "keypress"
}]
```



You can do so using `sepWith()` in `shuffleSequence`:

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

Putting everything together

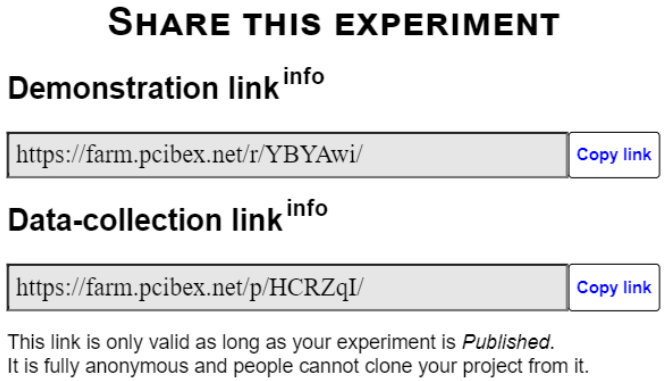
We now have a complete experiment with several **Message** controllers added.

```
// Defaults and other settings //

//// A message to show to participants at completion
var completionMessage = "Thank you for your participation!"

//// 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",
  "directions",
  startsWith("practice"),
  "setcounter",
```

Getting the experiment up

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

1. Go to the PCIbex farm website and log in.
2. Click into (or create) your experiment and edit the `main.js` file.
3. Click **Share** in the menu bar to the right and copy the **Demonstration link** (for testing and development) or **Data-collection link** (available after publishing)



Demo: https://farm.pcibex.net/r/YBYAwi

Published: https://farm.pcibex.net/p/HCRZqI

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 differences in *reading time*.

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 (e.g., directions).

"DashedSentence" Controller

The **"DashedSentence"** Controller creates self-paced reading trials.

We can simply replace **"acceptabilityJudgment"** with **"DashedSentence"** in `items`:

```
var items = [
  ...
  [["gp.trans",1], "DashedSentence", {s: "While Anna trained the kitten
  [["gp.intrans",1], "DashedSentence", {s: "While Anna dressed the kitte
  [["gp.trans",2], "DashedSentence", {s: "Since Dave improved the depart
  [["gp.intrans",2], "DashedSentence", {s: "Since Dave worried the couns
  ...
]
```

And specify appropriate settings for **"DashedSentence"** in `defaults`:

```
var defaults = [
    "DashedSentence", {
       mode: "self-paced reading",
       display: "dashed"
    }]
```

Our second experiment

Finally, after re-writing some of the "**Message**" items, we have a new experiment!

```
// Defaults and other settings //

//// A message to show to participants at completion
var completionMessage = "Thank you for your participation!"

//// 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 = [
    "DashedSentence", {
       mode: "self-paced reading",
       display: "dashed"
    }]


// Presentation Order //

var shuffleSequence = seq(
  "intro",
  "consent",
  "directions",
  startsWith("practice"),
  "setcounter",
  sepWith("sep", rshuffle(startsWith("gp"), startsWith("filler")))
)
```

Question and Form Controllers

Yes/No | Forced-Choice-1 | Forced-Choice-2 | Free-Response

You can use the **Question** controller to ask **Yes/No** comprehension questions:

```
[["YesNo_example",1],
 "Question",
 {q: "Was the kitten trained?", as: ["Yes", "No"]}]
```

Was the kitten trained?

1. Yes

2. No

Question and Form Controllers

Yes/No | **Forced-Choice-1** | Forced-Choice-2 | Free-Response

The **as** parameter sets the **answer choices** presented to participants.

```
[["FC1_example",1], "Question", {
  instructions: "Choose the more natural sentence.",
   as: ["Who did you see that ate bread?",
        "What did you see the girl who ate?"]}]
```

1. Who did you see that ate bread?

2. What did you see the girl who ate?

Choose the more natural sentence.

Question and Form Controllers

Yes/No | Forced-Choice-1 | **Forced-Choice-2** | Free-Response

You can ask for **continuations** by modifying the **q** parameter.

```
[["FC2_example",1], "Question", {
  q: "While Anna dressed the baby ______",
 instructions: "Choose the more natural continuation.",
 as: ["started to cry.", "he started to cry."]}]
```

While Anna dressed the baby _____

1. started to cry.

2. he started to cry.

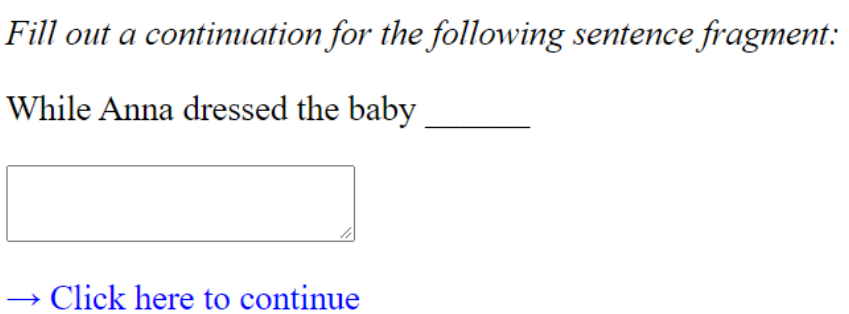
Choose the more natural continuation for this sentence.

Question and Form Controllers

Yes/No | Forced-Choice-1 | Forced-Choice-2 | **Free-Response**

Use **Form** controller with the **"textarea"** HTML tag to collect a **free response**:

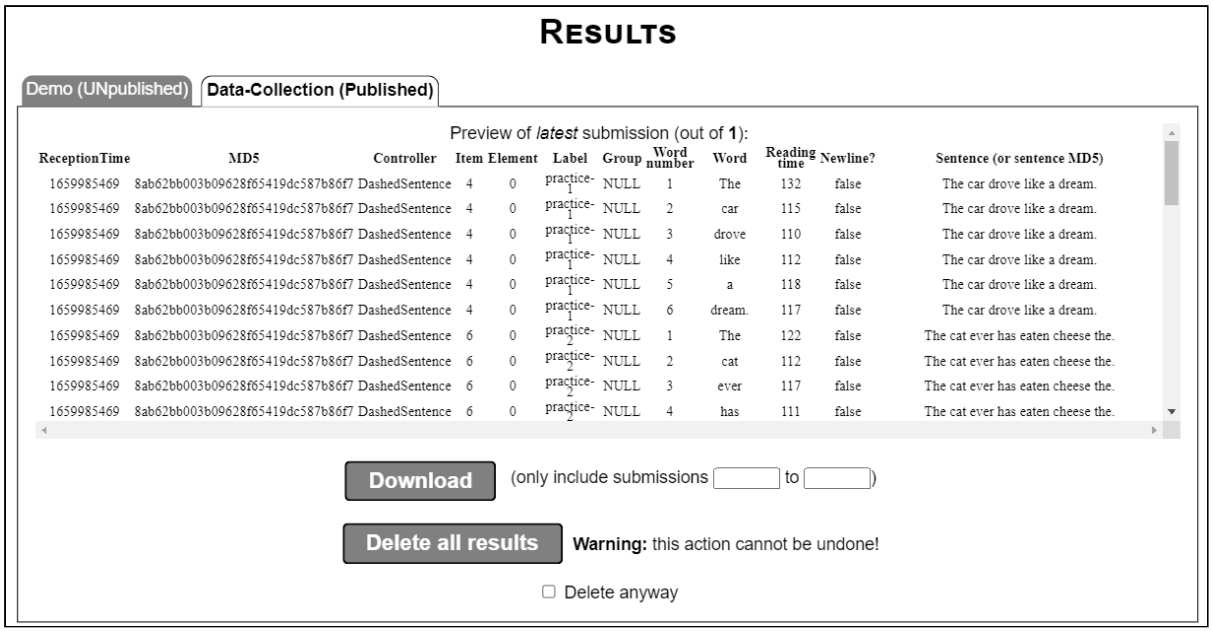
```
[["FR_example",1],
 "Form", {html: ["div",
   ["em", "Fill out a continuation for the sentence fragment:"],
   ["p", "While Anna dressed the baby ______"],
   ["textarea"]
]}]
```



3. Analysis of the results

Where to find the data

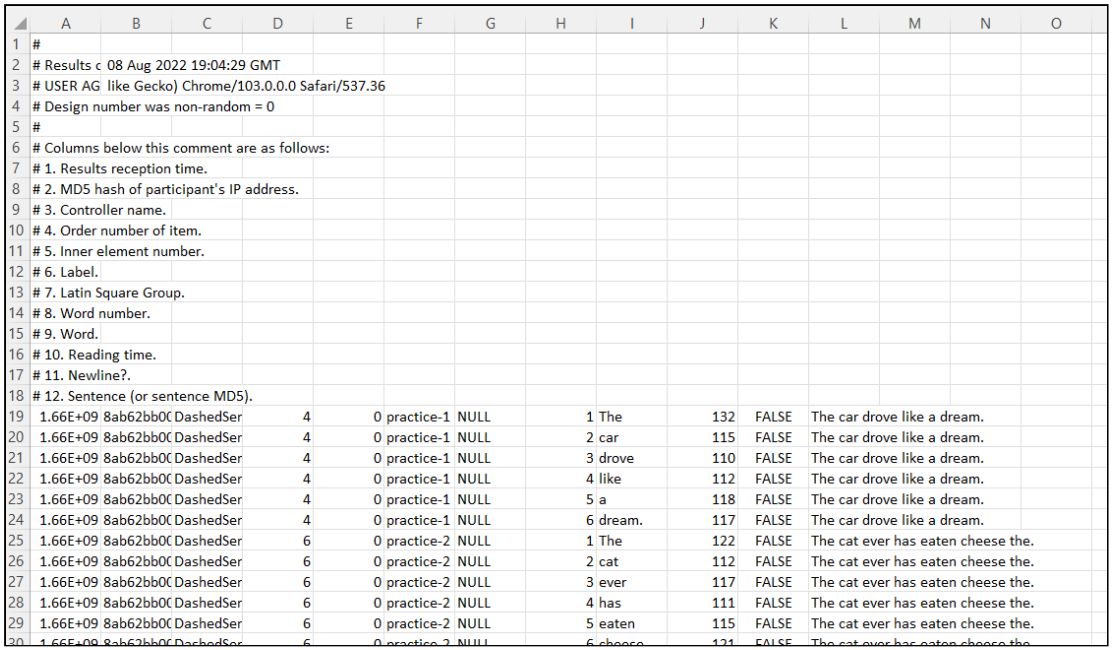
The data collected in the experiment can be found on the experiment page, in the **Results** tab to the right side of the dashboard.



It gives you a preview of the results and options to download or delete the data that's been collected so far.

The *results.csv* file

If you download the data, you get back a **csv** file called `results.csv`, meaning that each line contains a set of values that are separated by a comma.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	#														
2	# Results c	08 Aug 2022 19:04:29 GMT													
3	# USER AG	like Gecko) Chrome/103.0.0.0 Safari/537.36													
4	# Design number was non-random = 0														
5	#														
6	# Columns below this comment are as follows:														
7	# 1. Results reception time.														
8	# 2. MD5 hash of participant's IP address.														
9	# 3. Controller name.														
10	# 4. Order number of item.														
11	# 5. Inner element number.														
12	# 6. Label.														
13	# 7. Latin Square Group.														
14	# 8. Word number.														
15	# 9. Word.														
16	# 10. Reading time.														
17	# 11. Newline?.														
18	# 12. Sentence (or sentence MD5).														
19	1.66E+09	8ab62bb0(	DashedSer	4	0	practice-1	NULL	1	The	132	FALSE	The car drove like a dream.			
20	1.66E+09	8ab62bb0(	DashedSer	4	0	practice-1	NULL	2	car	115	FALSE	The car drove like a dream.			
21	1.66E+09	8ab62bb0(	DashedSer	4	0	practice-1	NULL	3	drove	110	FALSE	The car drove like a dream.			
22	1.66E+09	8ab62bb0(	DashedSer	4	0	practice-1	NULL	4	like	112	FALSE	The car drove like a dream.			
23	1.66E+09	8ab62bb0(	DashedSer	4	0	practice-1	NULL	5	a	118	FALSE	The car drove like a dream.			
24	1.66E+09	8ab62bb0(	DashedSer	4	0	practice-1	NULL	6	dream.	117	FALSE	The car drove like a dream.			
25	1.66E+09	8ab62bb0(	DashedSer	6	0	practice-2	NULL	1	The	122	FALSE	The cat ever has eaten cheese the.			
26	1.66E+09	8ab62bb0(	DashedSer	6	0	practice-2	NULL	2	cat	112	FALSE	The cat ever has eaten cheese the.			
27	1.66E+09	8ab62bb0(	DashedSer	6	0	practice-2	NULL	3	ever	117	FALSE	The cat ever has eaten cheese the.			
28	1.66E+09	8ab62bb0(	DashedSer	6	0	practice-2	NULL	4	has	111	FALSE	The cat ever has eaten cheese the.			
29	1.66E+09	8ab62bb0(	DashedSer	6	0	practice-2	NULL	5	eaten	115	FALSE	The cat ever has eaten cheese the.			
30	1.66E+09	8ab62bb0(	DashedSer	6	0	practice-2	NULL	6	cheese	121	FALSE	The cat ever has eaten cheese the.			

Lines that start with a pound symbol "#" are **comments** that include *metadata*.

The ones that start with numbers tell us what **variable** the columns correspond to.

The variables

All IBEX experiments return these **7** variables:

1. **Time** ("Results reception time")
2. **Participant ID** ("MD5 hash of participant's IP address")
3. **Controller** ("Controller name")
4. **Item number** ("Order number of item")
5. **Inner element number** ("Inner element number")
6. **Condition** ("Label")
7. **Item set** ("Latin Square Group")

The variables

In our case, we only care about **4** of these:

1. Time
2. **Participant ID**: A unique ID for each participant
3. **Controller**: The type of task/trial (e.g., AcceptabilityJudgment)
4. Item number
5. Inner element number
6. **Condition**: The condition label for the trial (practice-1, gp-trans, ...).
7. **Item set**: The item set number (1,2,3,...).

The last two variables **uniquely identify** the stimuli defined in `items`:

[["*Condition*", *Item set*], "*Controller*", {s: "*Sentence*"}]

[["gp.trans", 1], "DashedSentence", {s: "..."}]

The variables

You also get other variables depending on the **Controller**

For self-paced reading with **"DashedSentence"**, we get **5** more variables:

1. **Word number** - The index of the word in the sentence.
2. **Word** - The text of that word.
3. **Reading time** - Reading time for that word.
4. **Newline?** - 0 or 1 indicating whether there was a line break.
5. **Sentence** - The text of the sentence.

These are also outlined in the documentation for **"DashedSentence"**, so you know what variables to expect beforehand.

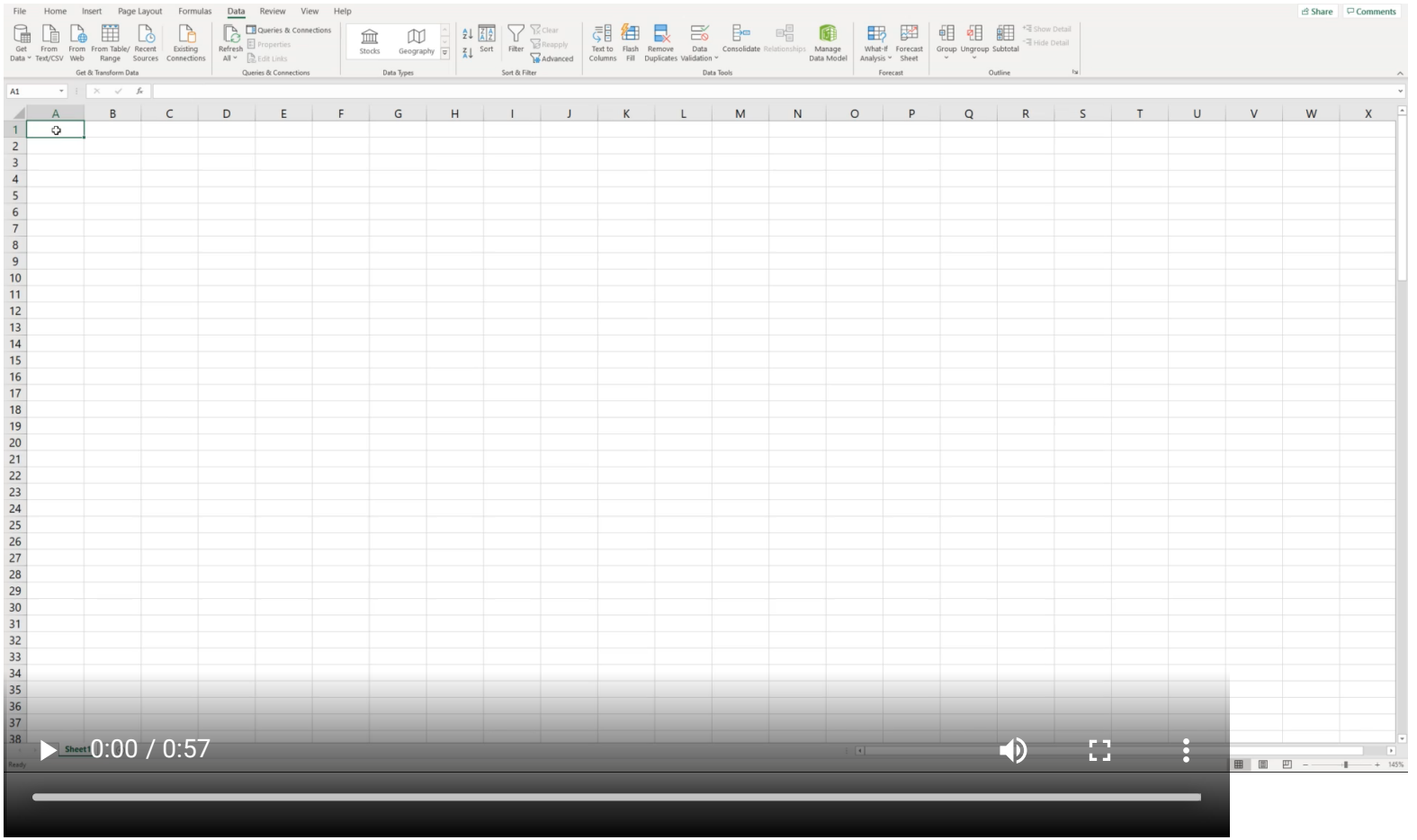
For an actual analysis of the results, we need the results to be imported somewhere as a **data frame**, preferrably in a rectangular table format where rows are observations and columns are variables.

Importing - Excel

Steps for importing the data into Excel:

1. Copy the text of the *results* file from IBEX
2. Paste into the first column of an Excel spreadsheet
3. Highlight that column and click **Data** tab -> **Filter**
4. Click on the dropdown arrow that appears at the top cell
5. Click **Text Filters** -> **Begins With...** and type in "#"
6. Go to **Home** tab -> **Find & Select**, check "Visible cells only", and click OK
7. Right click on any part of the sheet and select **Delete Row**
8. Click the first column again and go to **Data** tab -> **Text to Columns**
9. Check "Delimited", "Comma", then "General" for each prompt
10. Add an empty row at the top and manually type in the column names from the comments of the original *results* file

Importing - Excel



Complex results

Some Controllers return multiple lines of results, with differing number of columns

For example, each **AcceptabilityJudgment** trial is a combination of **Message** and **Question** controllers under the hood, so it logs two lines of data:

	A	B	C	D	E	F	G	H	I	J	K
1	#										
2	# Columns below this comment are as follows:										
3	# 1. Results reception time.										
4	# 2. MD5 hash of participant's IP address.										
5	# 3. Controller name.										
6	# 4. Order number of item.										
7	# 5. Inner element number.										
8	# 6. Label.										
9	# 7. Latin Square Group.										
10	# 8. Sentence (or sentence MD5).										
11	1.66E+09	52feb2564	Acceptabil	4	0	practice-1	NULL	The car drove like a dream.			
12	# 8. Question (NULL if none)..										
13	# 9. Answer.										
14	# 10. Whether or not answer was correct (NULL if N/A).										
15	# 11. Time taken to answer..										
16	1.66E+09	52feb2564	Acceptabil	4	0	practice-1	NULL	NULL	5	NULL	1433

In more complex cases like these, using a **script (R/Python)** is recommended

Importing - R

Steps for importing into R:

1. Download the `results.csv` file from PCIbex
2. Open it with the `read_pcibex()` function from the read_pcibex.R script

DashedSentence | AcceptabilityJudgment

```
source("read_pcibex.R")
results <- read_pcibex("SPR_results.txt")
```

```
# A tibble: 6 × 5
  Controller_name Label      Word_number Word   Reading_time
  <chr>           <chr>            <dbl> <chr>         <dbl>
1 DashedSentence  practice-1           1 The             132
2 DashedSentence  practice-1           2 car             115
3 DashedSentence  practice-1           3 drove           110
4 DashedSentence  practice-1           4 like            112
5 DashedSentence  practice-1           5 a               118
6 DashedSentence  practice-1           6 dream.          117
```

Importing - R

Steps for importing into R:

1. Download the `results.csv` file from PCIbex
2. Open it with the `read_pcibex()` function from the read_pcibex.R script

DashedSentence | AcceptabilityJudgment

```
source("read_pcibex.R")
results <- read_pcibex("Acceptability_results.txt")
```

```
# A tibble: 6 × 4
  Label      Latin_Square_Group Sentence_or_sentence_MD5   Answer
  <chr>      <chr>              <chr>                      <dbl>
1 practice-1 NULL               The car drove like a drea…    NA
2 practice-1 NULL               <NA>                           5
3 gp.intrans 2                  Since Dave worried the co…    NA
4 gp.intrans 2                  <NA>                           6
5 gp.trans   1                  While Anna trained the ki…    NA
6 gp.trans   1                  <NA>                           5
```

Break

Exercise

Task

The experiment script incomplete.js has some missing pieces.

```
//////////////////
//// Exercise ////
//
// There are a total of EIGHT tasks for you to complete in this script.
// The instructions for each task are written out as comments.
// Some parts of the task are completed for you and your job is to fill in the blanks ______.
//
// Make sure to consult the IBEX manual! -
https://github.com/addrummond/ibex/blob/master/docs/manual.md
//
//////////////////


// Task #1: Send a completion message that says "Thank you for your participation!"
var ______ = ______

var showProgressBar = false

var defaults = [
    "AcceptabilityJudgment", {
       as: ______, // Task #2: Make the acceptability judgment scale from 1-5
       presentAsScale: true,
       instructions: "Use number keys or click boxes to answer.",
       leftComment: "(Bad)",
       rightComment: "(Good)"
    }]
```

Task

Directions:

1. Download incomplete.js
2. Create a new experiment on your PCIbex account called "WorkshopExercise" and replace the contents of *main.js* with *incomplete.js*.
3. Follow the directions in the script to fill in the blanks.
4. Complete as many of tasks as possible (**8 total**).

If you finish, save the edits, refresh, and **preview** the experiment in the bottom panel or click **Open in new tab**. Check to see that your experiment looks similar to the complete version - https://farm.pcibex.net/p/ZvbOXD.

Make sure to reference the IBEX documentation!

https://github.com/addrummond/ibex/blob/master/docs/manual.md

What about the PCIbex framework?

For several of the standard experimental paradigms in linguistics, the PCIbex framework **directly ports** controllers from IBEX. So for tasks like self-paced reading, you get the same experiment from a different style of code.

You can clone the **PCIbex version** of the workshop's SPR experiment here - https://farm.pcibex.net/r/nYsovX

PCIbex has great support for experiments involving multimedia (audio, video, etc.). The PCIbex documentation is a good place to get started - https://doc.pcibex.net

IBEX is always good to know due to the sheer amount of linguistics experiments already written in IBEX.

PCIBex is not difficult to pick up after IBEX, should you choose to learn it.

Thank you!

Many thanks to...

- Brian Dillon (UMass Amherst) for permission to use materials from the LSA Minicourse "Doing Experiments for Linguists" (Brian Dillon & Rodica Ivan)
- Jon Sprouse (UConn) for permission to use materials from LSA 2016, Experimental Syntax Workshop
    - Link to Jon's course on Introduction to Experimental Syntax Methods: https://sprouse.uconn.edu/courses/experimental-syntax
- Florian Schwarz and Jeremy Zehr for permissions to use materials from their PCIbex workshops.
- Attendees of the 2021 BK Winter School Workshop on Experimental Linguistics/Syntax at Sungkyunkwan University for valuable feedback on the workshop materials.