Appendices to "A streamlined approach to online linguistic surveys"

Michael Yoshitaka Erlewine[1] **• Hadas Kotek**[2]

Posting a linguistic survey on AMT involves, in AMT jargon, the creation of a *Project*. Each Project has a corresponding HTML template with a fixed structure and number of items. The *Requester* uploads a *Batch* file, which we have referred to in this paper as a randomized lists file. Each row in the Batch file corresponds to one randomized list of experimental items and will be treated by AMT as a single Human Intelligence Task (HIT). In the AMT interface, the number of participants and predicted costs of a study are determined *per HIT*.

Below we present detailed information about the use of the different components of *turktools*, which were introduced in section 4 of the paper. These tools as presented in the order that they are used in the recommended workflow, described in the paper. Further ongoing documentation for the *turktools* project can be found online at: http://turktools.net/use/.

A. *Skeletons* and *Templater*: Creating an HTML template for use on Mechanical Turk

Once a research question is formulated and an experiment type has been selected to test it, the presentation of the survey can be determined. Surveys are presented on AMT using an HTML template. We recommend creating a template in two steps: (a) choosing an appropriate *skeleton* out of the ones offered as part of *turktools* and editing it to fit your study, and then (b) creating a template from the skeleton using the *Templater*.

Skeletons are recipes for different experiment types that abstract away from the number of items that will be presented in a given study. They are stripped-down versions of HTML templates which should be edited to fit each particular design or experiment. We recommend the use of a programming text editor to edit these files.[3] For each experiment type we surveyed in Section 3 of the paper, we provide a corresponding skeleton:

(1) **Supplied skeleton files:**

a.	`binary.skeleton.html`	grammaticality judgment, yes-no task
b.	`slider.skeleton.html`	grammaticality judgment, gradient judgment task
c.	`likert.skeleton.html`	grammaticality judgment, Likert scale task
d.	`magnitude-estimation.skeleton.html`	grammaticality judgment, magnitude estimation task
e.	`constant-sum.skeleton.html`	grammaticality judgment, constant-sum scale task
f.	`image-choice.skeleton.html`	picture selection task
g.	`binary-image.skeleton.html`	picture judgment, yes-no task
h.	`binary-audio.skeleton.html`	audio stimuli, yes-no task[4]
i.	`completion.skeleton.html`	word-completion task
j.	`sentence-choice.skeleton.html`	sentence-completion task
k.	`completion-menu.skeleton.html`	completion task with drop-down menu

[1] National University of Singapore, Singapore

[2] McGill University, Montreal, Canada

[3] Good, free options include *Notepad++* for Windows and *TextWrangler* for Mac.

[4] For technical details on the use of the audio template, see http://turktools.net/use/audio.html .

A skeleton is itself an HTML file, but contains *substitution tags* which will be filled in by the *Templater*. These tags are all wrapped in double curly braces, i.e. `{{…}}`. The skeletons we provide all share the same basic structure, illustrated in Figure 1 below, using the `completion.skeleton.html` skeleton. The very top of the skeleton has a comment to remind users that a template must be created out of the skeleton before uploading to AMT.[5] Next are the instructions for the survey, including any practice items, as well as a consent statement and contact information for the experimenters.[6] Finally, there is text that requests workers to turn on JavaScript in their browser. This enables a counter that is included in all the skeletons which helps workers make sure that they have answered all the questions in the survey.[7]

Notice the substitution tags in this skeleton. The *Templater* will replace `{{code}}` with the experiment's unique code, and `{{total_number}}` with the number of items presented in the experiment.

{{! This is a template skeleton; use templater.py! }}

Survey Code: {{code}}

PLEASE COMPLETE AT MOST ONE {{code}} SURVEY. YOU WILL NOT BE PAID FOR COMPLETING MORE THAN ONE SURVEY WITH THIS CODE.

Instructions
This questionnaire presents {{total_number}} English sentences. Each sentence contains a gap and there are two options below the sentence for what should go in that gap. Choose the option that sounds more natural to you. Here is an example:

There is ___ boy in the room.
○ the ○ a

In this example...

Consent Statement: ...

If your browser has JavaScript turned on, a counter will be displayed at the bottom of the page indicating how many questions have been answered. **It is highly recommended that you turn on JavaScript and use this tool before submitting to ensure that all questions have been answered and you can receive payment.**

{{#items}}
{{number}}. {{field_1}} ___ {{field_2}}
○ {{field_3}} ○ {{field_4}}

{{/items}}

Demographic questions, questions about native language.

0 questions (out of {{total_number}} total) have been answered. If you submit now, you will not be paid.

After submitting this HIT, do NOT submit another HIT with survey code {{code}}. You will not be paid for completing more than one survey with this code.

Figure 1. An example skeleton file for a completions study, `completion.skeleton.html`

[5] Comments, which will be completely ignored and removed by the *Templater*, are tags with an exclamation point, as in `{{!...}}`.

[6] Approval for your proposed experiments from the Institutional Review Board (IRB) at your institution may be required before you run them on AMT. Please consult your institution's IRB.

[7] Researchers can make the use of JavaScript obligatory, for example if JavaScript is used for critical manipulations such as hiding and revealing stimuli or recording timing information. For most of the template skeletons that we provide, it is possible for participants to take part in surveys even if JavaScript is not enabled in their browser. The result is that they will not see the counter, but other functionality of the survey will be maintained.

The "survey code" at the top of the template is useful for record-keeping purposes on the experimenter's end. It is also used to instruct workers to only complete one survey of a certain type. This is often desired in linguistic experiments, as collecting multiple judgments from the same participant may result in skewed results of a study, if the researchers mistakenly treat these observations as independent. The same effect can be achieved, for example, by color-coding experiments instead of, or in addition to, displaying experiment codes, which will make it easier for participants to remember whether or not they have already participated in the study. Another way to ensure that workers only participate in a study once is to add JavaScript code to the template to check the current Worker's ID against a list of previous participants' IDs and to block the Worker from participating if they have already done so.[8]

The main body of the skeleton is the items block, beginning with `{{#items}}` and ending with `{{/items}}`. The items block contains one sample item of the shape that all items in the survey will take. When a template is created out of the skeleton, this block will be duplicated as many times as there are items in your survey. The item contains the tag `{{number}}`, which will be replaced with the item number in the experiment, as well as a number of `{{field_n}}` tags. In this case, each item is made up of four fields: fields 1 and 2 are the parts of the sentence before and after the gap and fields 3 and 4 correspond to possible answer options. Each of these `{{field_n}}` tags will be replaced by a different "field" in the items file. The makeup of the items file will be described in the following section.

Below the items block are some demographic questions. We always ask participants to indicate their native language and any other languages they speak, although we clarify that payment is not contingent on their answers to these questions. Researchers may wish to add questions about gender, age, origin, etc. Finally, at the bottom of the skeleton is a counter to help workers ensure that they reply to all questions before submitting their survey. Below the counter is text warning participants not to accept a second survey with the same code after submitting their survey.

Note that all parts of the skeleton can be edited, moved around or removed to suit the needs of the researchers and the study. We recommend saving the changes to the skeleton in a dedicated folder that will contain all files pertaining to the same experiment. At least some changes will only have to be made once—for example, the consent statement should be the same in all surveys conducted under the same IRB approval.

After choosing a skeleton and editing it, a template must be created out of the skeleton using the *Templater*. The script *templater.py* should be saved in the same folder as the skeleton. The *templater.py* script and all other Python scripts described in this paper can be run in the following manner:

(2) **Executing a Python script (here *templater.py*):**
- a. UNIX shell: move (`cd`) to the directory that contains the script and execute `python templater.py`.
- b. Mac OS X: Right-click on the file in the Finder and choose Open With… > Python Launcher.
- c. Windows: Double-click on the file.[9]

The *Templater* will ask for three pieces of information:

(3)
- a. the skeleton file name: one of the skeletons provided here or others you create yourself.
- b. the number of items in your survey: including all experimental and filler items (but not practice items which are coded in the skeleton).
- c. a survey code: any letter-number combination you choose.

Once these are provided, a template file will be created. The template file will be located in the same folder as the skeleton file and the *templater.py* script.

[8] These alternative methods are not at the moment a default part of *turktools*. See the online documentation for how to implement these features in your skeleton, at http://turktools.net/use/color.html and http://turktools.net/use/check.html. We thank two anonymous reviewers for suggesting these methods.

[9] For Mac OS X and Windows, these instructions assume that Python was installed using the standard binary packages provided at http://python.org. If not, use the UNIX shell instructions via the Terminal on Mac, and open using the `python` application on Windows. Please consult the latest information at http://turktools.net regarding supported Python versions and dependencies.

B. Creating an items file

Once a design and a template have been decided on for your survey, items need to be created. For a simple 2×2 experiment crossing two *factors* (here: *definiteness*, *there-construction*) with two *levels* each (here: +/– *definite*, +/– *there-construction*), each target item set will contain four different *conditions*. A sample target *item set* for a sentence judgment task will look as in (4).

(4) **A sample item set:**

	Stimulus:	*Condition name:*	*Factor values:*
a.	A boy is in the room.	`indef-no.there`	*–definite, –there-condition*
b.	The boy is in the room.	`def-no.there`	*+definite, –there-condition*
c.	There is a boy in the room.	`indef-there`	*–definite, +there-condition*
d.	There is the boy in the room.	`def-there`	*+definite, +there-condition*

Turktools uses a naming convention whereby the *condition name* specifies the chosen value for each of the factors, separated by hyphens (5). Names of this form will be assumed by the *Decoder*, which we will discuss in Appendix F.[10] For our *there*-construction experiment, this results in conditions names as in the middle column of example (4), where the settings of the two factors are shown in the third column and the first column shows the actual sentence that we would like participants to rate.

(5) **The structure of a condition name:**

```
<factor 1 value>-<factor 2 value>-<factor 3 value>-…
```

An experiment is composed of different *sections*, where normally at least one section will be used as *fillers*, and the others will be *targets*. This designation is made when using the *Lister* (Appendix C). Each *item* in the items file has two parts, each beginning on a new line, as described in (6) below:[11]

```
# target 1 indef-no.there
A boy is in the room.

# target 1 def-no.there
The boy is in the room.

# target 1 indef-there
There is a boy in the room.

# target 1 def-there
There in the boy in the room.
```

Figure 2. Sample item set with four conditions, in *Lister* format (repeated from Figure 3 in the paper)

[10] It is possible to use other naming schemes for condition names. In such a case, however, part of the decoding process, which is automated in the *Decoder*, will have to be done manually by the researchers. See footnote 18.

[11] This structure is very similar to that used for items files in *Linger*, written by Doug Rohde (http://tedlab.mit.edu/~dr/Linger/), and Gibson et al.'s (2011) *Turkolizer*. Note that our terminology differs slightly: Gibson et al refers to a group of *trials* together forming an *item*, which corresponds to our notions of individual *items* which are grouped together into *item sets*.

Furthermore, the structure of the item body in our *Lister* format is much more flexible, allowing for an arbitrary number of fields. Unlike in *Linger* and Gibson et al.'s *Turkolizer*, comprehension questions are not given a special status but are instead treated like any other field that could be added to an item. If a comprehension question is given or if some filler items are expected to have correct answers, these values should be specified as *hidden fields* in the items file; see discussion following (6).

(6) a. **The item header**. The item header consists of four components, separated by spaces.
   (i) the symbol `#`;
   (ii) the name of the *section* (e.g., `target` in the example in Figure 2, above);
   (iii) the item set number within the *section* (we recommend 1, 2, 3, …);
   (iv) the *condition* name (e.g., `indef-no.there`, `def-no.there`, `indef-there`, `def-there` in our example). The condition names are not constrained in any way, but it is most helpful to use labels that reflect the experimental design, and in particular to describe the setting of the *factors* of interest for that item.

   b. **The item body**. The item body consists of *fields*, each on its own line.[12] Line *n* corresponds to the text that will be substituted for the text `{{field_n}}` in the skeleton file. A field may specify a sentence to be judged, choices for completions, a picture, audio file, a context, a comprehension question, etc.

Note that the choice of experimental design and skeleton will dictate the format of your items. That is, each item must have at least as many fields as there are fields in the sample item in the skeleton that you have chosen. It is possible to add *hidden fields* to the body of an item by adding extra lines that do not have corresponding substitution tags in the HTML template. This is a way to specify additional information for a given item that is useful for the researchers but should not be available to the participants. For example, if a skeleton refers to `{{field_1}}` through `{{field_4}}` but not `{{field_5}}`, then anything that occurs in the 5th line (and onward) of an item's body will not be displayed on the screen and participants will not have access to this information. However, the data will still be part of the results file created by AMT after the experiment is completed and can be used as part of the data analysis. For example, the correct answers to comprehension questions or expected answers to filler items may be specified in this way.[13]

The format for a sample item set in a *there*-construction study is given in Figure 3 below. For illustration, the experiment here pairs each sentence with a picture. Each item consists of two fields, the first specifying the sentence and the second, a picture.[14] This corresponds to two substitution tags in the `binary-image.skeleton.html` skeleton: `{{field_1}}` stands in for the sentence, and `{{field_2}}` for the URL in an `<img>` tag, specifying the picture to be shown.

```
# target 1 indef-no.there
A boy is in the room.
http://DOMAIN/experimentpictures/definiteness-effect/picture1.png

# target 1 def-no.there
The boy is in the room.
http://DOMAIN/experimentpictures/definiteness-effect/picture1.png

# target 1 indef-there
There is a boy in the room
http://DOMAIN/experimentpictures/definiteness-effect/picture1.png

# target 1 def-there
There in the boy in the room
http://DOMAIN/experimentpictures/definiteness-effect/picture1.png
```

Figure 3. Sample item set with four conditions in *Lister* format

[12] If a line break is required within a single text field, use the HTML code `<br/>` .

[13] If some items have more than one hidden field, it is important to make sure that each field is exclusively used for one purpose (e.g. field 5 for the correct answer to the item and field 6 for the expected answer to a comprehension question). It is possible to leave a field empty if a given item does not have a value associated with some field (e.g., no expected correct answer to an item, but there is an expected correct answer to the comprehension question—leave field 5 empty, as a blank line, and enter the expected correct answer to the comprehension question in field 6).

[14] Resources such as images and audio or video files must be hosted separately, rather than included with the AMT experiment. We recommend hosting such resources on your own web space, for example in a hidden directory on your website, and including links to those resources in the items.

C. *Lister*: Creating randomized lists of items to be uploaded on Mechanical Turk

The *Lister* takes an items file as input and returns a comma-separated-value (`.csv`) format file with randomized lists of items that can be uploaded to AMT. AMT will combine this item lists file with the template created by the *Templater* to create the final survey that will be viewed by participants. When you run *lister.py*, the program will maximally ask for the following information:

(7)
a. the name of your items file (described in the previous section)
b. which sections should be treated as fillers
c. how many filler items you would like placed between each target item
d. how many filler items you would like placed at the beginning and end of the experiment
e. how many lists you would like to create
f. whether or not you would like the reverse of each list to also be created, to help reduce any ordering effects that may occur

After this information is given, the *Lister* will create as many lists as requested, using a Latin Square design. That is, for each item set in a given section, an item for only one of the specified conditions for that item set will be included in a list. If the number of conditions does not match for all item sets in a section, an error will be given. Note that if all the item sets in a particular section have only one item, they will all be included, regardless of the condition names given.[15]

It is possible to dictate a minimum number of filler items to be placed between each two target items and at the beginning and end of each list, if there are enough filler items. The algorithm creates counterbalanced and randomized lists for each section individually (including the fillers), and then spreads fillers between target items to minimally comply with the constraints given to the *Lister*. If additional filler items are left after this process, they are randomly distributed across the list. If there are not enough filler items to comply with the constraints given to the *Lister*, an error message will appear. Note that it is possible to have fewer filler items than target items, but in that case it is not possible to impose restrictions on their position relative to target items. It is also possible to have no sections designated as fillers. In this case, all the items in the list will be randomized first within their sections and then across other sections, but no other conditions can be imposed on the ordering.

If some items have more fields than others (for example, if some items have expected correct answers but others don't), the *Lister* will produce a message notifying the user of the discrepancy. The user may choose to go back and check that the file is correct or to continue with the current file. Assuming no such issues, the *Lister* will output a randomized lists file in CSV format (`xxxxxxx.turk.csv`) that can be uploaded onto AMT. For example, if the input to the *Lister* is the items file `definiteness-effect.txt`, the output will be named `definiteness-effect.turk.csv`. The randomized lists file contains a row for each randomized list, indicating the actual text that the workers will see, with a header line at the top. AMT will substitute these strings into the HTML template for the participants to see.

D. *Simulator*: Verifying that the survey works properly

With your items randomized into lists and your HTML template prepared, you are in principle ready to upload your survey onto AMT. However, before doing so, we recommend pre-testing your study using the *Simulator* script provided here. The *Simulator* takes as input a randomized lists file (`xxxxxxx.turk.csv`) and a template and creates a version of the survey based on one of the lists in the item lists file (you can choose which list to simulate). The *Simulator* allows researchers to bypass the rather cumbersome process of uploading a survey onto AMT, or

[15] In a filler section where each item set is made up of one item each, it may be helpful to use different condition names even though all items will be displayed. This may help make the analysis script easier to create. For example, some fillers may be designated as 'catch' items. Accuracy can then be calculated for those items only and then used to discard participants from the analysis.

using AMT's *Sandbox* to simulate an experiment.[16] The output of the *Simulator* is an HTML file that can be opened in any web browser.

It is beneficial for researchers to complete their own study, for several reasons. Doing so helps spot any typos or errors in the rendering of the survey. If your items file includes hidden fields, you should make sure that all hidden fields are not displayed and all other fields are displayed as expected. Additionally, it is important to get a feeling for how hard the experiment is and how long it takes to complete it. This will help you determine payment for your survey. Furthermore, because of the existence of *satisficing* behavior on AMT—the behavior of some workers to put in minimal effort and still get paid—it is useful to make sure that there are no easy strategies to complete the study without doing the linguistic task of interest. Strategies for dealing with such behavior include the addition of comprehension questions to items, the inclusion of 'catch' filler items that should have clear correct answers, and the use of diverse types of fillers that cannot all be answered using the same strategy.

E. Uploading surveys onto Mechanical Turk and downloading results files

After verifying that the survey is ready, it can be uploaded onto AMT. We refer the reader to our online instructions on uploading surveys onto AMT: http://turktools.net/use/turk.html. In some cases it may be necessary to reject submissions from workers who have completed more than one survey. It may also be desirable to reject workers with low accuracy on comprehension questions and 'catch' filler items and those workers who failed to complete at least a certain proportion of the study, e.g. 80%. To quickly identify workers who completed more than one survey, open the file in software such as Excel and sort by 'WorkerId.' Some suggestions and sample code for additional exclusion criteria and how to identify workers who meet them using R can be found in the analysis script `analysis.r` which is provided with *turktools* and can be modified by researchers to meet their own needs. In order to maintain a positive reputation as a requester on AMT, it is advisable to approve submissions in a timely fashion and to maintain a low rejection rate.

A note on being a good requester: We believe it is important to maintain a positive reputation as a requester and to treat workers fairly. Note that workers expect their HIT results to be accepted unless something has gone seriously wrong, for example if they did not complete a large portion of the experiment or exhibited obvious guessing behavior. Workers strive to maintain a low rejection rate. This is very important as it allows them to work on tasks that restrict workers' acceptance rates. As a result, many workers will avoid HITs by requesters who reject many HIT submissions, and will be very upset at being rejected if they believe that they completed the task in good faith. To maintain a good relationship with workers, we suggest that new requesters make sure to write clear instructions for their surveys which include specific rejection criteria. We also encourage requesters to respond to email inquiries from workers promptly.[17]

As a general good practice, we recommend accepting all submissions unless a worker clearly did not follow the instructions. We have provided code that requesters can use to restrict the access of workers who they have identified as bad experimental participants (see http://turktools.net/use/check.html). It is also possible to restrict tasks to workers who have a certain acceptance rate and a certain number of tasks completed, using AMT's "worker requirements" field, when creating a template online. We recommend using these criteria rather than using Masters, since Amazon's commission fee is lower for non-Masters and as far as we can tell, Masters are not better linguistic consultants than other AMT workers.

[16] Amazon Mechanical Turk's Sandbox allows researchers to upload "dummy" versions of their experiment, much like using the *Simulator*. However, to use the Sandbox it is necessary to set up the experiment independently on the Sandbox. This entire process must be repeated every time a mistake is discovered. Then one must go through all the same steps once again in order to create the actual survey on AMT; there is no way to transfer a survey from one platform to the other.

[17] AMT workers maintain websites to keep track of requesters' reputations. Among other things, they collect information about payment, rejection rates, overall fairness, and promptness in responding to inquiries. See for example http://turkopticon.ucsd.edu/.

F. *Decoder*: Preparing the results for analysis

After your survey is completed, download a final copy of the raw results file from the AMT *Manage* tab. We recommend re-naming this file, e.g. `definiteness-effect.results.csv` (to continue using the same example as above), and saving it in the same folder as the other documents produced in the course of preparing your survey. Use the *Decoder* script, which should be in the same folder. The *Decoder* will ask for the name of the raw results file (here: `definiteness-effect.results.csv`) and return a decoded file, here named `definiteness-effect.results.decoded.csv`.

Unlike the raw results file, the decoded results file contains one row for each *item* in each list in the survey. For each item, the decoded file specifies information about the `Section`, `Condition`, `Item`, `List`, `PresentationOrder`, and all fields and participant responses associated with that item, including hidden fields. In addition, the *Decoder* adds a number of `Factor` columns, corresponding to different factor values. These factor values are constructed by simply splitting the `Condition` value up by hyphens.[18] For example, in our example where an item may have `Condition` value '`indef-no.there`,' `Factor1` would be set to '`indef`' and `Factor2` would be set to '`no.there`.' This processing of condition names using the schema in (5) facilitates the analysis of the data later.

Metadata about a submission, including the `AssignmentId` (unique for each worker-survey pair), `SubmissionStatus`, `WorkerId`, `WorkTimeInSeconds` (for the entire survey) and answers to demographic questions, is duplicated for each item for a given worker in the file. Therefore, for each participant who worked on your survey there will be as many rows in the file as there were items in the survey,[19] and the meta-information about the submission will be the same in all rows from that submission.

Once decoded in this way, the results are ready to be analyzed by any standard statistics software. We have provided some basic code in the analysis script `analysis.r`, but researchers should adapt the script for their own needs.[20]

G. Hosting AMT-style experiments on your own server with *turkserver*

In addition to posting an experiment on AMT, the tools we have provided with this paper can be used to generate surveys to be hosted on the researchers' own server. In order to do so, we have developed a program called *turkserver*, freely available at http://turktools.net/use/server.html under a liberal open-source license. *Turkserver* requires a server capable of serving PHP scripts, and is designed for Apache web servers. See the online documentation for detailed instructions on installation and usage.

To use this option, construct the experiment as described in Appendix A-D. An experiment is set up with *turkserver* by uploading the template and item lists file (`.turk.csv`). A URL for the experiment is generated, and this URL can then be shared with potential participants. *Turkserver* handles the random assignment of participants to different lists, displaying the survey using the chosen list of items, and recording the results in a format compatible with AMT's results files.

There are some differences between running an experiment on AMT and on *turkserver*. AMT comes with many potential participants registered on the system, and therefore acts as a subject recruitment tool as well. On the other hand, the potential participant pool is limited to those people who have chosen to sign up to be a worker on AMT. With *turkserver* it is up to the experimenter to recruit subjects and send them to the experiment URL. Similarly, unlike AMT, *turkserver* does not handle payment to participants. *Turkserver* does, however, attempt to produce a

[18] If condition names were not constructed using factor names and hyphens, as suggested in (5), custom code may be needed in the analysis script to recover factor values from the value of the `Condition` column.

[19] But note that if a worker completed two surveys, for example, there will be twice as many rows. All the rows corresponding to the first survey completed by this worker will share metadata pertaining to the first submission and all rows corresponding to the second survey will share metadata pertaining to the second submission. The `AssignmentId`s will be distinct, but the `WorkerId` will be the same across these rows. This allows us to easily identify workers who completed multiple surveys. At the time of writing, AMT cannot itself enforce that each worker only complete one survey per logical experiment.

[20] See also some analysis suggestions on the *turktools* website at http://turktools.net/use/analysis.html.

unique `AssignmentId` for each submission, as AMT does, and to produce a `WorkerId` for each participant that is stable across experiments.

Running AMT-style surveys on *turkserver* can be useful for situations where the AMT subject pool does not include many speakers of a particular language, but such speakers can be recruited separately. It is also ideal for when the experimenter wishes to recruit participants from a particular group, such as a vetted subject pool, a class, or among colleagues. Furthermore, the ability to generate experiments that run both with and without AMT offers great flexibility. For example, one could use the exact same experimental materials (template and item lists file) and conduct the experiment with the general public on AMT, and also on recruited non-naïve (working linguist) participants via *turkserver*, and compare their behavior.