Pyhab User Manual

v0.8

Summer 2019

Dr. Jonathan F. Kominsky

1. What is PyHab?

Infant and toddler looking-time studies primarily rely on manual looking-time coding software like jHab and xHab, which are old, opaque, and not open-source. PyHab is a script written for PsychoPy, a free, open-source python-based stimulus presentation software (with which the author is not affiliated), that fully replicates the capabilities of xHab and jHab while adding several features.

The biggest advantages of PyHab over its predecessors are that it can be used to directly control stimulus presentation, and it's free and open-source. xHab and jHab needed one person coding looking times and a second person controlling the stimuli, and communicating when to end one trial and advance to the next, and when to advance from habituation trials to test trials. This introduces logistical hassle and imprecise timing. PyHab can present movie files or animated stimuli and advance trials and trial types appropriately based on live looking time coding. One person can run an entire experiment. It's similar to Habit2, but open-source and with slightly different capabilities.

You might worry that this means you can't run these studies with a blind coder. Not at all! Provided that the coder can't determine what the participant is seeing or hearing (which will just depend on your setup), PyHab can be configured to tell a coder only when a trial starts and ends, and use a masked condition file so that the coder only needs to input a code and not know what condition is being presented. In habituation designs when the test trial can come after a variable number of habituation trials, PyHab doesn't even need to tell the coder when the test trial happens, in some ways making it *more* blind than other programs can achieve!

If you want to be kept up-to-date on new versions, please join the Pyhab announcements mailing list: https://groups.google.com/forum/#!forum/pyhab-announcements

2. Getting PyHab

- You need PsychoPy, first of all. PyHab is a script that runs in a PsychoPy environment. For PyHab version 0.7.2 and later, you will need **at least** version 3.0.6, and you will want the standalone .dmg file for Mac or a .exe file for Windows. You can download it here: https://www.psychopy.org/download.html As of this writing, Psychopy3 version 3.1.2 is the most recent and best build to use.
- **You DON'T need to know how to program!** There is a builder interface that should allow you to build and conduct studies with no programming at all. For people who do know Python, the code is documented on ReadTheDocs: https://pyhab.readthedocs.io/en/latest/
- To use PyHab for stimulus presentation, you will also need a computer with a (minimum) two-screen setup. One screen will be your stimulus presentation screen, which will display stimuli to your participant. The other will be the coder's screen, which will display a small graphical interface telling you when a trial begins and ends. The displays cannot be mirrored, you need to have an extended desktop (like you would for a slide show with presenter view).
- If you want to present movie files, you will want your stimuli in .mov, .mp4, or .m4v format for the most reliable performance. PsychoPy's movie presentation system works with other formats, but I have not tested all of them with PyHab and I'm not sure how well they work. Some formats work better on Mac than windows.
- In addition, **you will want to design movies that have about two to four frames of buffer at the end**. This is because of the way that PsychoPy plays movies. If a movie file gets to the absolute end (i.e. past the last frame), the movie is unloaded from memory. In order to loop movies without having to reload the movie every time, I had to basically force PyHab to stop about 2 frames before the end of the movie (this

scales to the frame rate of the movie as of 0.7.2) to guarantee that the movie doesn't unload itself.

- You will need to install VLC media player on any computer that you plan to use to present stimuli. VLC ensures that the right codecs are available on your system for playing movie files. Fortunately, VLC is completely free and can be found here: https://www.videolan.org/vlc/index.html
- **On Windows**, you will need ffmpeg installed to be able to use the stimulus presentation features. If you attempt to run a study with stimuli and get an imageio crash, follow these directions: In PsychoPy, open a coder window, then at the bottom select the tab labeled "shell". At the prompt in this tab, type "import imageio" and hit return, then type "imageio.plugins.ffmpeg.download()" and hit return. This should download and install ffmpeg for PsychoPy. If it fails, close PsychoPy and then run PsychoPy as an administrator (right click > run as administrator).
- **Also on Windows**, you will want your movie files to be relatively small. For unknown reasons, MoviePy will use a prohibitive amount of memory to load larger movie files. You may find that you need to export them using VLC in windows to get access to these settings. If your experiment hangs on "loading movies", this is why. Macs, for some reason, do not have this problem (or at least have it to a lesser degree).
- **Citing PyHab:** If you use PyHab in your experiment, **please cite both it and PsychoPy**. The relevant citations are:

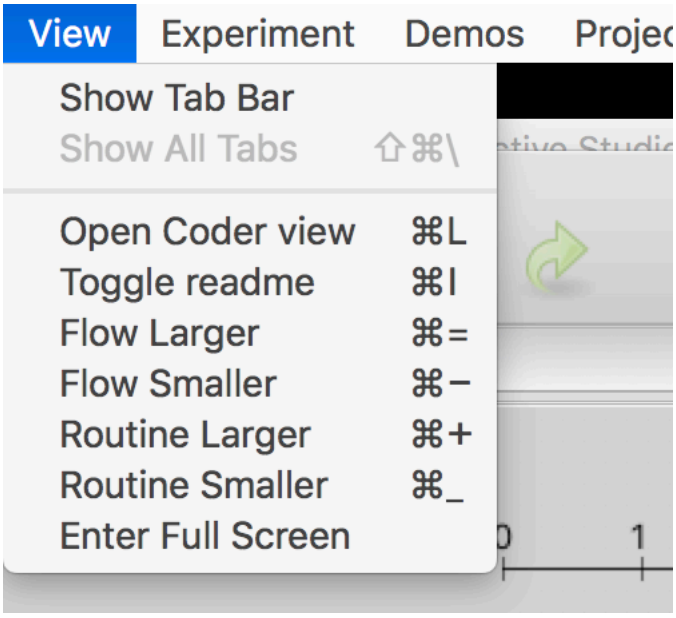
Kominsky, J. F. (2019) PyHab: Open-Source Real Time Infant Gaze Coding and Stimulus Presentation Software. *Infant Behavior & Development, 54*, 114-119. doi:10.1016/j.infbeh.2018.11.006

Peirce, J. W., Gray, J. R., Simpson, S., MacAskill, M. R., Höchenberger, R., Sogo, H., Kastman, E., Lindeløv, J. (2019). PsychoPy2: experiments in behavior made easy. *Behavior Research Methods.* doi:10.3758/s13428-018-01193-y

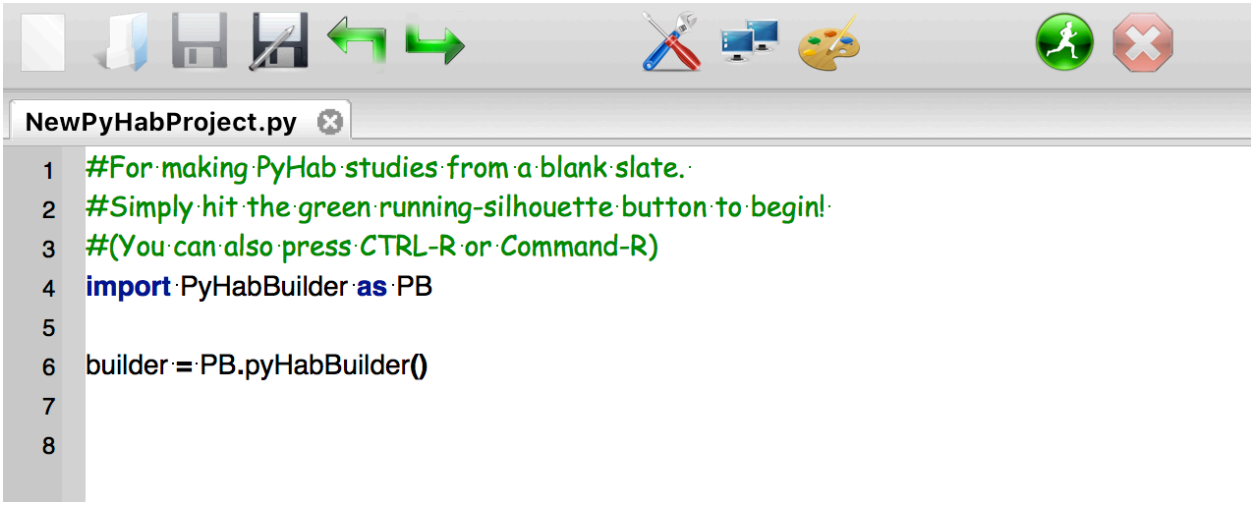
Installation instructions

Assuming you have PsychoPy already installed, all you need to do is go to https://github.com/jfkominsky/PyHab/releases and click "Source code (zip)". This will download a .zip file called PyHab-master, which you can then unzip into a folder. This folder has everything you need to get started. Put it wherever you want, but don't modify anything inside the folder or move anything into or out of it.

Once you've got your PyHab folder where you want it, open PsychoPy in coder view.



Then go to file > open and find the script called "NewPyHabProject.py" in the PyHab-master folder. This will open a very, very short script in the coder window. All you need to do is hit "Run" (the green running man icon) to launch the PyHab builder (see Chapter 3).



If you want to see a PyHab study in action, from the coder, go to file > open, find the PyHab-master folder, the open the "PyHabDemo" folder within it, and finally select "PyHabDemoLauncher.py". This will open a somewhat longer script. Again, just hit the "Run" button, and this will open a dialog that will allow you to either run the demo experiment (see Chapter 4) or open its builder view to see how it is put together.

3. How to build your study in PyHab

a. Relevant concepts

Let's define a few important terms:

- **Launcher:** The script you run in an experiment's folder initially, which gives you access to both running studies and editing them in the builder.
- **PyHab Builder**: A rudimentary GUI that allows you to construct your experiment. This is not to be confused with the (much, much superior) PsychoPy builder. Rather, this is something I built for PyHab specifically that is a little less refined but better-suited to the specific requirements of designing infant looking-time studies, and particularly habituation studies. Future updates may better integrate it into the PsychoPy builder.
- **Experimenter window**: The window presented to the experimenter running the study, which (optionally) includes information about the current trial and the live status of the coding keys.
- **Stimulus window**: The window on which the stimuli appear, if PyHab is being used for stimulus presentation. What the baby sees.
- **Trial type**: PyHab builds studies in terms of trial types, each with its own label. Each label is enclosed in single quotes and is usually text (e.g., 'A'). Avoid symbols like \ or / or %, which can confuse Python in various ways (especially \). PyHab also reserves certain characters (. and ^), and will not let you name a trial with either.
    - There is exactly one special trial type that changes the behavior of the program: 'Hab'. The 'Hab' trial type will create a series of habituation trials that will go until a habituation criterion is met (see Habituation). If you aren't running a habituation study, don't call any of your trial types 'Hab', but literally any other series of letters should be fine.
- **Block**: A block is a sort of 'meta' trial-type made up of other trial types or blocks. They are partially recursive. You can have a block made up of other blocks, but you can't ever

have a block that includes itself, which would loop infinitely. PyHab is smart enough to stop you from making a block that includes a block that includes itself, too.

    - You can also make 'Hab' a block, in order to fit multiple trial types into a habituation trial. This is useful if, for example, you want to present a video followed by a still-frame, as each one will be its own trial type.

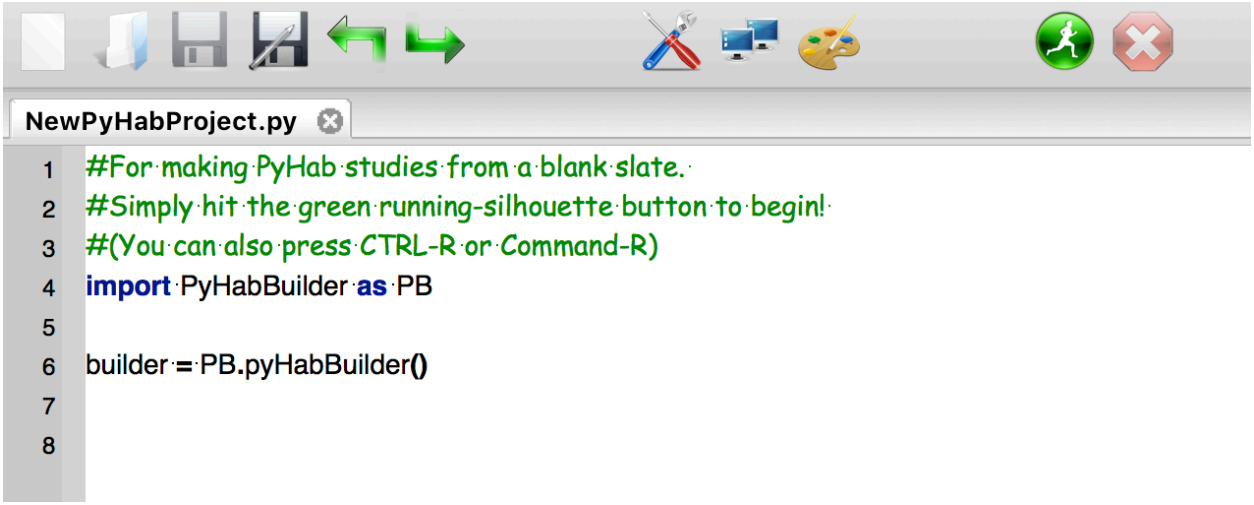
- **Stimulus file/stimuli**: PyHab expects all stimuli to be in the form of movie, audio, or image files, which you import into your PyHab experiment from the builder. You can also run PyHab just as a looking-time coding software with no stimulus presentation, in which case no stimulus files are needed.
- **Stimulus library**: When you add media files to PyHab, you create a "library" of files which can then be associated with different trial types.
- **Attention-getter**: A short sound and/or animation designed to attract an infant's attention to the stimulus screen.
- **Primary coder and Secondary coder**: When to advance through trials, PyHab *only* pays attention to the primary coder. The secondary coder is someone coding at the same time for determining reliability, but is completely optional.
- **Verbose data vs. Summary data**: Summary data is a single file that gives one line per trial, and some summary statistics for that trial. Verbose data is a data file where each line is one gaze on or gaze off. Verbose data is much more granular and is necessary for calculating certain forms of inter-rater reliability. Each coder gets their own verbose data file.

    - There are also habituation summary data files and block summary data files. These files compress multi-trial blocks into one line, whereas the default summary data file (which is always saved, even if other summary data files are saved) is always one line per trial.

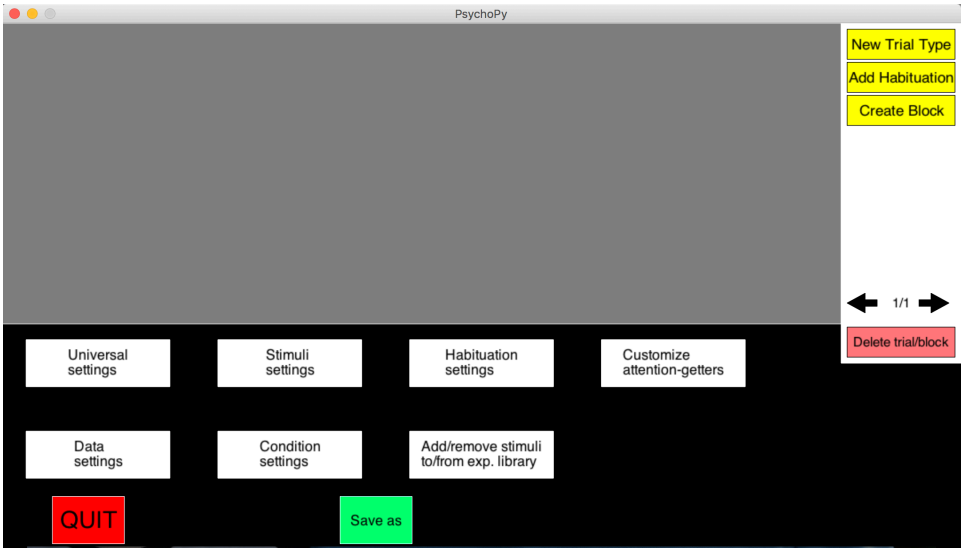
- **Preferential looking version**: PyHab behaves slightly differently when you are running a preferential looking study. The "normal" PyHab only allows you to code whether they are looking at the screen or not. The preferential looking version does not allow for multiple coders and does not compute reliability (but reliability can be computed after the fact).

b. Creating your first experiment

Open the "NewPyHabProject.py" script in the PsychoPy coder window (see Chapter 2). You will see something like this. Hit the green running man icon on the right to launch the builder window.



This may take a few seconds to start up, but a new window should open that looks like this.



This is your main builder view, and what you will be dealing with most of the time. The top area with the grey background is the study flow, which will ultimately show you the order in

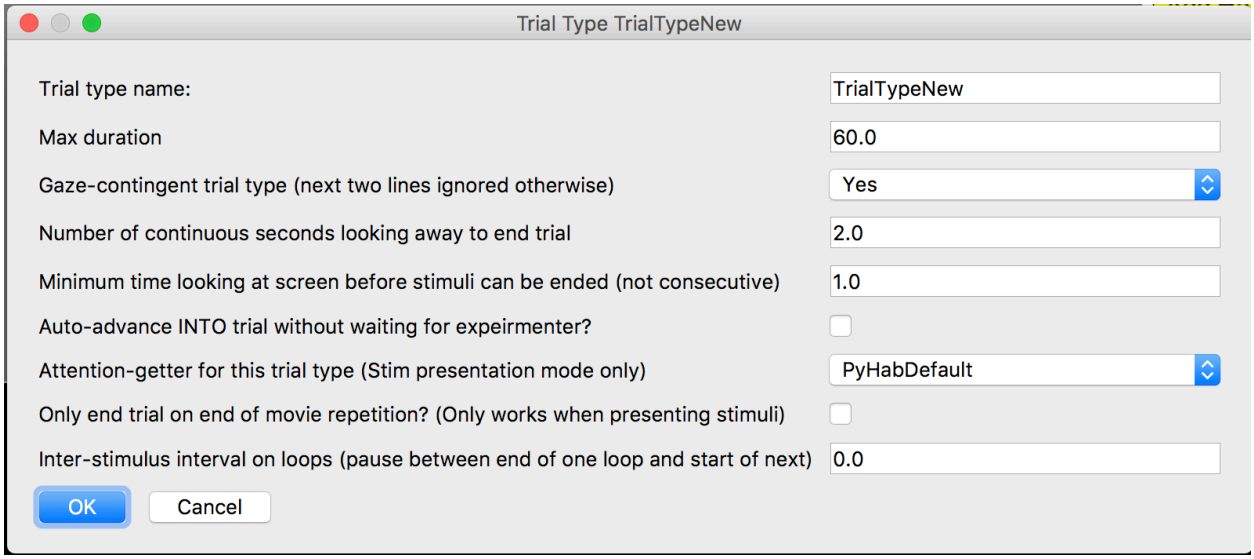
which your trials will be presented. The area on the right is the trial types palette, where you can define new trial types for your study, edit existing trial types, or delete trial types. All of the other settings can be accessed through the six buttons in the lower half of the screen.

Each of the buttons here will open up a separate dialog box to let you change the settings. Note that, on Windows, the builder window will close while the dialog box is present. This is intended behavior. It will reopen when you click “OK” or “Cancel” in the dialog. On Mac, the builder window will remain open, just in the background.

Trial types and blocks

The first thing to do is create a trial type. In the white area in the top-right corner of the builder window, there are four buttons: “New Trial Type”, “Add Habituation”, “Create Block”, and at the bottom, “Delete Trial Type.”

When you click “New Trial Type”, a window will appear that looks like this:



The top line is the name of your trial type. “TrialTypeNew” is a default, you can replace it with whatever you want (with some exceptions, but PyHab will let you know if a trial type name has problems). The second line sets the maximum duration of the trial, in seconds. By default this is set to 60 seconds.

The next line is a drop-down menu that lets you set whether the trial type is gaze-contingent, which has three settings. If this is set to “no”, the trial will run until its max duration, every time, regardless of infant’s looking behavior. If it is set to “OnOnly”, the trial will run until the maximum duration or the “Minimum on-time” criterion has been met. In other words, this is for a trial you want to end after the infant has looked at it, but regardless of whether they look away. If it is set to “Yes”, it will end either at the max duration or when the minimum-on AND look-away criterion are met.

The minimum-on and maximum-off criteria for the trial type are set in the next two lines. If the trial is not gaze-contingent, these values are ignored. If it is set to “OnOnly”, the first value

is ignored. The numbers are given in seconds. By default, the infant must look at the screen for at least one second, and then the trial will end when they look away for two consecutive seconds or the maximum duration has been reached, whichever comes first.

After that is a check box determining whether you want the experiment to auto-advance into that trial. PyHab's default behavior is that each trial must be started manually. In stimulus presentation studies this typically means pressing a key to present the attention-getter and then starting the next trial. If this box is checked, PyHab will instead automatically begin trials of this type after ending the previous trial, as soon as the ITI in the universal settings has passed. If you also turn off the built-in attention-getter (see stimulus settings), then this will automatically start playing the next movie as well.

Following that is a drop-down list of attention-getters, so you can select which one you want to use for that trial. The default attention-getter is a looming, spinning yellow rectangle accompanied by a rising musical scale. I've found it to be pretty effective. You can also add a customized attention-getter that you have made. If you do not want an attention-getter, simply select "None". Note that if you do not have an attention-getter, you won't be able to pause between the attention-getter and the rest of the stimulus, it will just start playing the stimulus as soon as you start the trial. The attention-getter also affects coding when you are not using PyHab for stimulus presentation: PyHab extracts the duration of the attention-getter and will emulate a delay of that length when coding off-line.

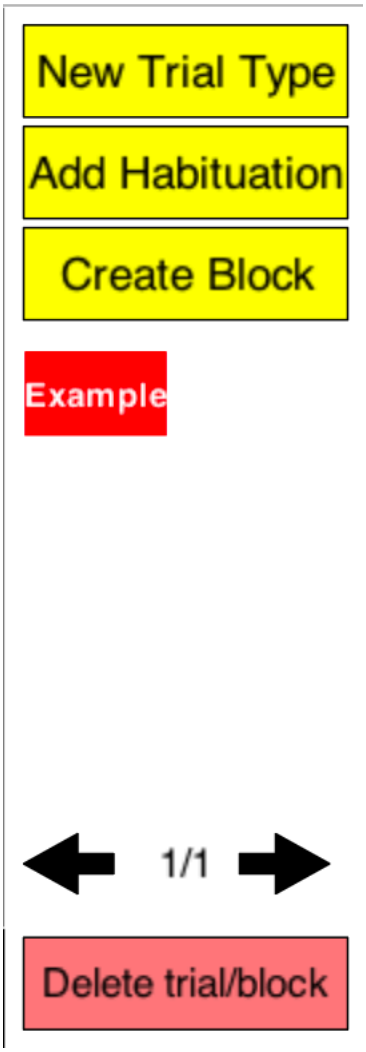
The next check-box is a little complex. Say you are using a movie or audio stimulus, and you don't want the movie to end anywhere in the middle, but instead, once some criteria have been met you want the trial to end at the end of the media's current loop. In that case, you would check this box. **However, this can make the timing of your trials different depending on whether you are running in stimulus presentation mode or not**. This is the only setting that can really do that. If you aren't using PyHab to control your stimulus presentation, or using still images as stimuli, this check-box does nothing.

> **Tip:** If you want a non-gaze-contingent trial to play a movie exactly once and then end, check this box and set the "max duration" value to less than the length of the movie. This checkbox takes precedence: It will play to exactly the end of the movie and then end the trial.

Finally, ISI between loops is for when you want to enforce a pause between loops of your stimuli during a trial. Decimals are allowed. If you are *not* using PyHab for stimulus presentation, this is used to determine the timing of a trial (so if the stimuli have an ISI, you can make sure your timing lines up by putting that value here).

Once you have named your trial type and set the options, hit "OK". This will add the trial type to your trial type palette on the right. At current, PyHab supports a maximum of twenty distinct trial types. Note that these are not the number of trials in the experiment, just the number of different *kinds* of trials. There is no hard limit on the number of trials in your experiment (though after 40 they will stop rendering in the study flow).

Now your trial type palette will look something like this:



The arrows and numbers at the bottom are page numbers. If you have more than eight trial or block types, they will spill onto a second page. Simply click the left and right arrows to go back and forth between pages.

From here, there are three things you can do:

- Left-click on the name of the trial type to add an instance of that trial type to the study flow.
- Right-click on the name to edit the trial type, like changing its maximum duration or whether it is gaze-contingent, or to remove stimuli from it (adding stimuli uses a different button).
- Click the "Delete a trial type" button to remove the trial type from the palette (and all instances of the trial type from the study flow).

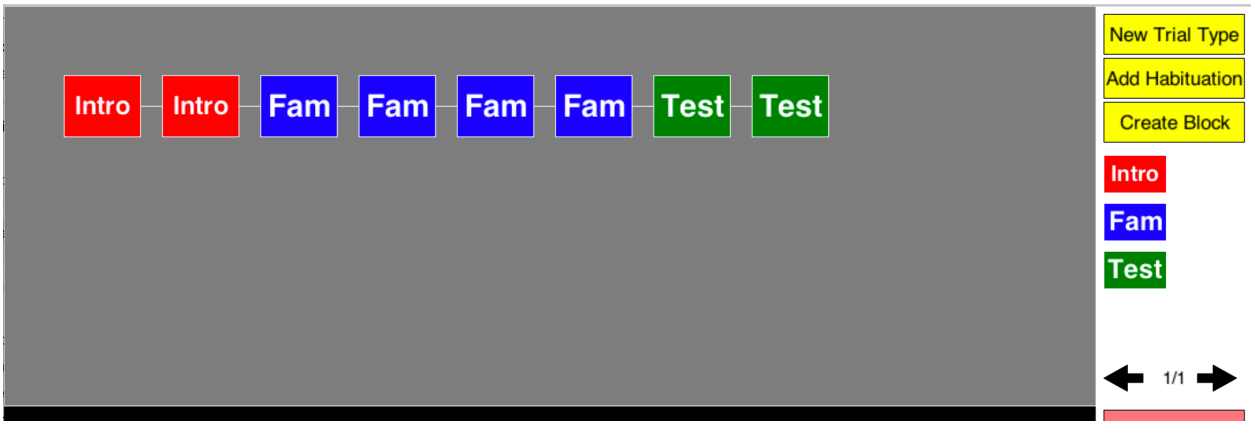
We'll go through each of these in turn.

Making your study flow

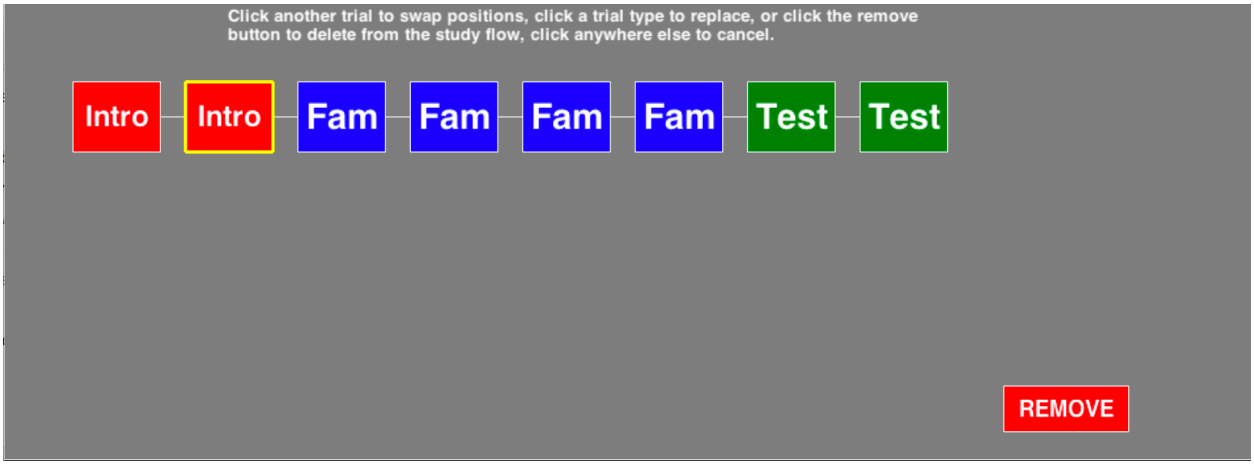
The "study flow" is the order in which different trial types will be presented. Say you have three trial types, intro, familiarization, and test. Your palette might look something like this.

Now say you wanted participants to go through two intro trials, four familiarization trials, and two test trials. You would click the intro button twice, the "fam" button four times, and the

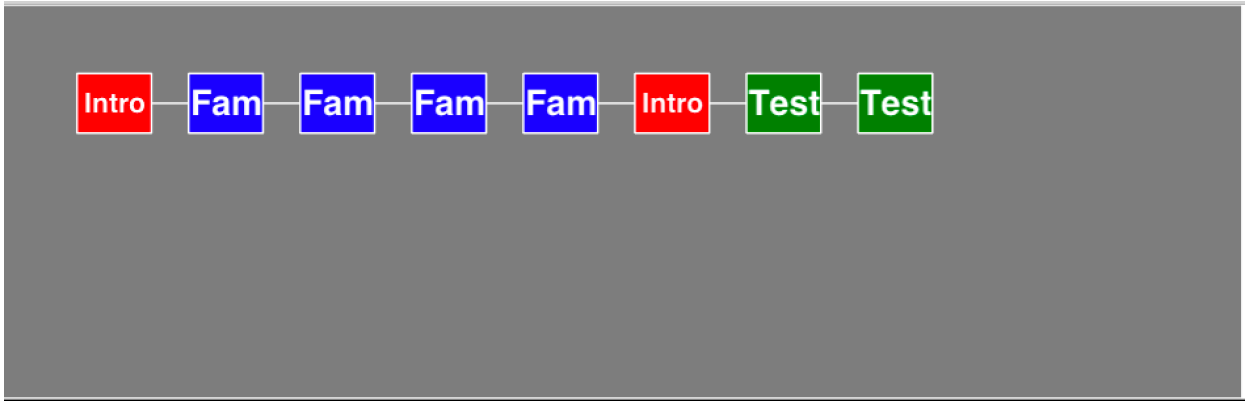
"test" button twice. This would leave you with a study flow that looked like this.



To move trials around in the study flow, replace them with a different trial type, or to remove them altogether, simply click on the trial in the flow. Say you wanted to move the second intro trial so that it was between the last familiarization trial and the first test trial. First, click on the second intro trial, which will highlight it, like so.



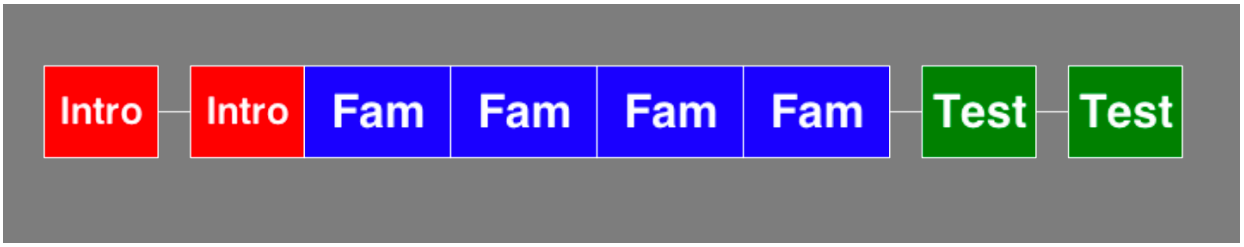
The instructions at the bottom are pretty self-explanatory. To move the intro trial to after the last Fam trial, simply click on the last Fam trial and they will switch places.



You can also swap a trial in the study flow with a different trial type by selecting it and then clicking the other trial type in the trial type palette. So, for example, if you selected “Intro” and clicked “Fam” in the trial type palette, the Intro trial would be turned into a Fam trial.

PyHab can render up to 40 trials in the study flow. After it hits this limit, it will render 39 trials and a counter of how many additional trials are not being rendered.

PyHab also marks auto-advancing trials in the study flow. Say “Fam” is an auto-advance trial, meaning that as soon as the trial before it finishes, it will start. To indicate this, instances of “Fam” in the study flow will now be immediately adjacent to the trials preceding them, except at the start of the next row in the flow (i.e., trials 1, 11, 21, or 31).

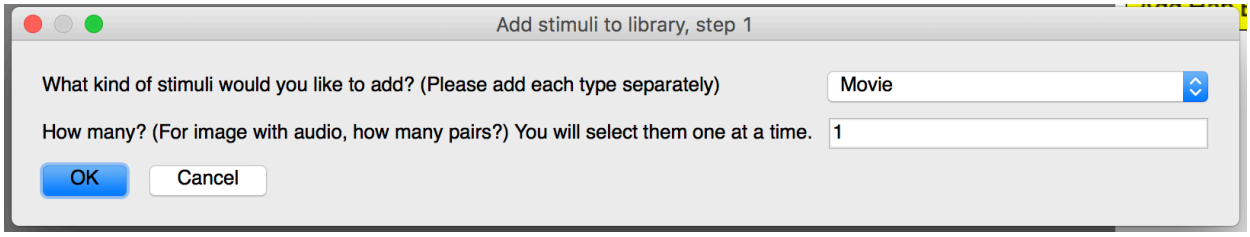


Note that this is ALL you can do with the study flow directly. If you want to change the order of stimuli within a trial type, or change the settings of a trial type, all of that has to be done in their respective menus.

Adding stimuli to the experiment library

In PyHab, the basic logic of the stimulus system is that you add stimuli to an internal experiment library, and then assign stimuli from the library to different trial types.

When you click the “Add/remove stimuli to/from exp. library” button, you will see the following dialog:

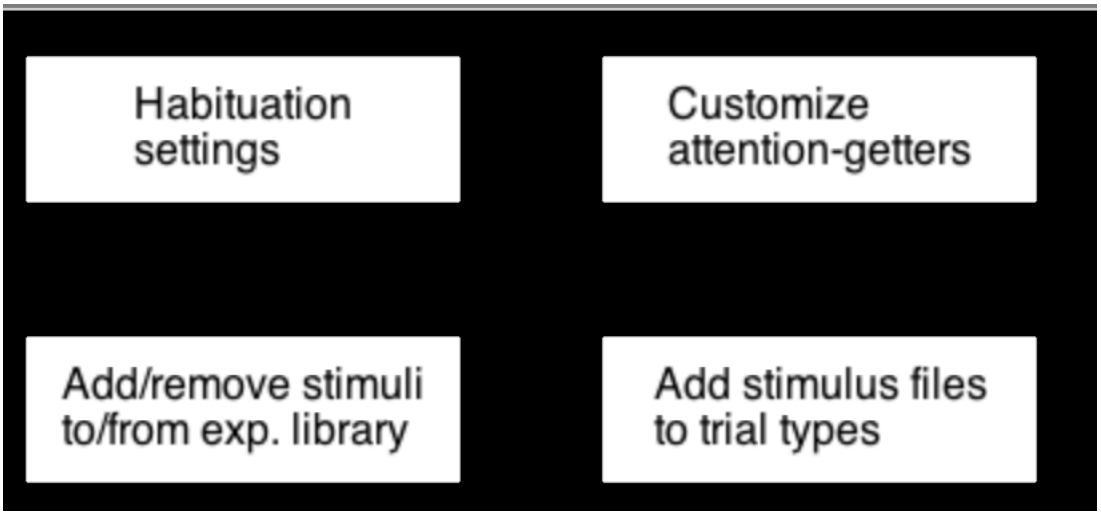


The first line lets you select what kind of stimuli you want to add. The second line specifies how many stimuli of that type you want to add. You can only add one type of stimuli at a time, but you can add as many of that type as you want. There are four types of stimuli you can use in PyHab:

- **Movie files**: On Mac, .mov files with MPEG-4 or H.264 encoding seem the most reliable. On everything else, .m4v and .mp4 files seem to work most of the time, but on Windows you might have trouble with movies that are too long (this is an issue with PsychoPy and really with Python itself, not PyHab-specific). Other file formats should work in theory, but the way Python plays movies is a little temperamental, so you may just need to try different file types and find out what works (and let me know!).
- **Images**: Virtually any common image format will work. Animated GIFs may not play, however. For any kind of animation, you will probably want a movie file.
- **Audio**: Most audio formats should work. PyHab will not be able to modify the file in any way, however, it will just play it as-is.
- **Images with audio**: If you want a still image with an audio file rather than a movie, PyHab can do this. These are treated as a separate category of stimuli in PyHab, treating each one as an image/audio pair.

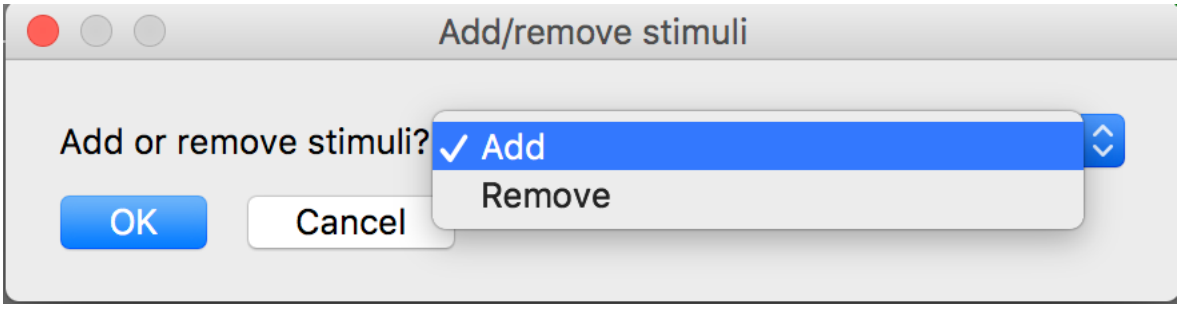
When you hit “OK”, you will then select each file, **one at a time**. For everything but “images with audio”, you will just see the normal “file open” dialog as many times as you specified for how many you want to add. For “images with audio”, you will get extra pop-ups telling you when you are selecting the audio file, and when you are selecting the image file.

When you are done, you will have added a bunch of stimuli to your library. This won’t be visible to you, except that there will be an additional button on the main screen, in the bottom-right.

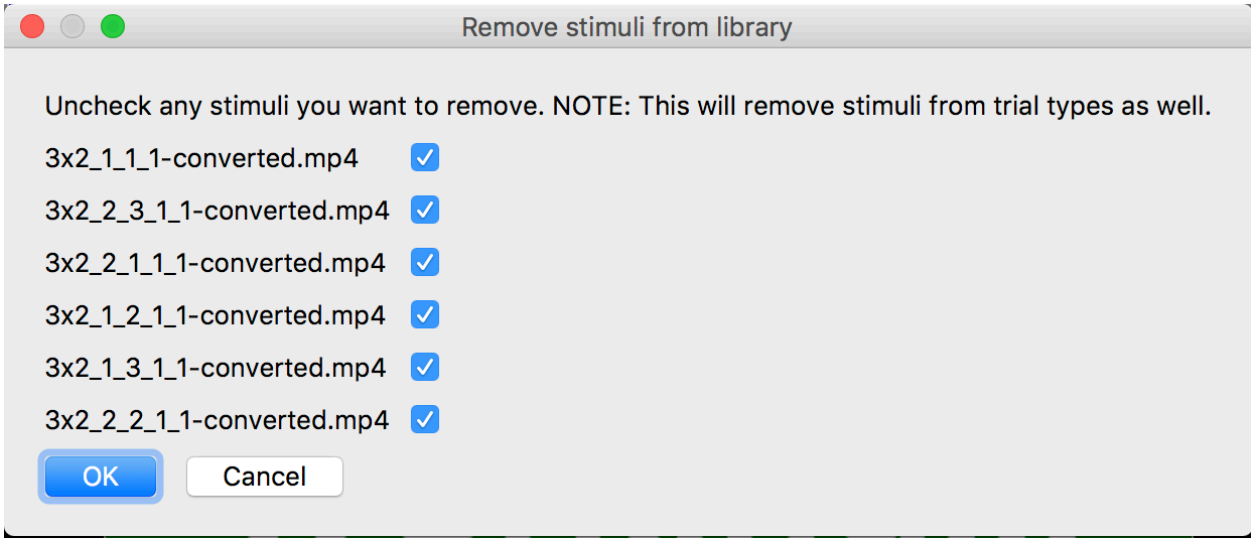


This button allows you to assign stimuli from your library to different trial types, and also add images that will display at the start and end of the experiment.

To remove stimuli from the experiment library altogether, simply click the add/remove stimuli button again. This will bring up a dialog that first lets you choose whether you want to add or remove stimuli.



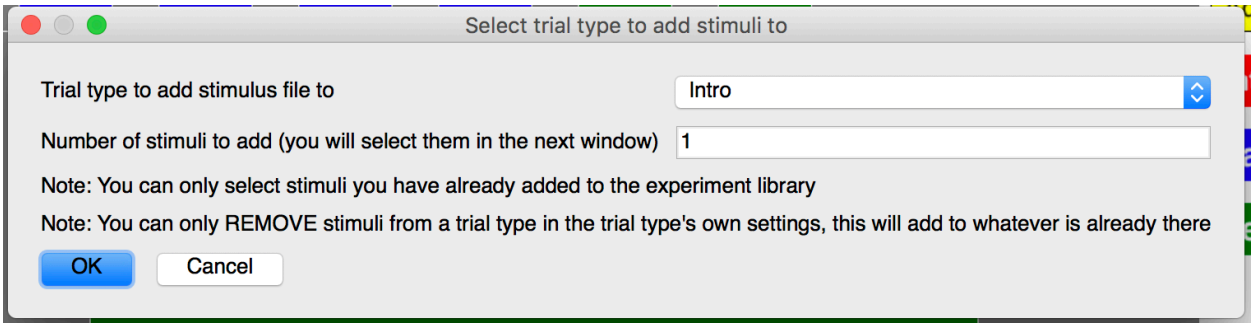
If you select ‘remove’, you will see a dialog like the one below, with a list of all of the different stimuli you have added to your experiment.



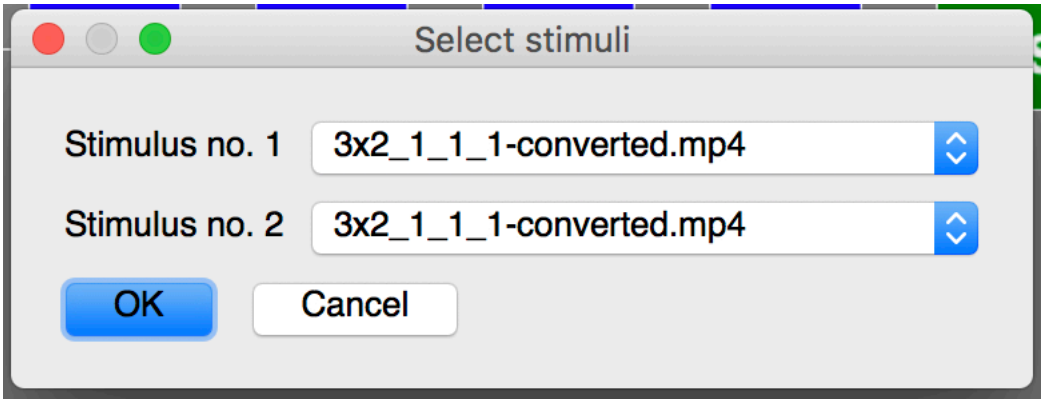
Uncheck any stimuli you want to remove and hit ‘OK’. This will remove the stimuli from the experiment library and from any trial types those stimuli have been associated with. Next time you save the experiment, it will also delete the corresponding stimuli files from the experiment folder (ONLY those in the experiment folder, it won’t delete the files from anywhere else).

Assigning stimuli to trial types

You need to create your trial types before you add stimuli to them. Let’s look at the demo experiment. Say we’ve added the six movie files to the experiment library, and we’ve created three trial types, “Intro”, “Fam”, and “Test”. Now, when you click “Add stimuli to trial types”, you will see the following window:

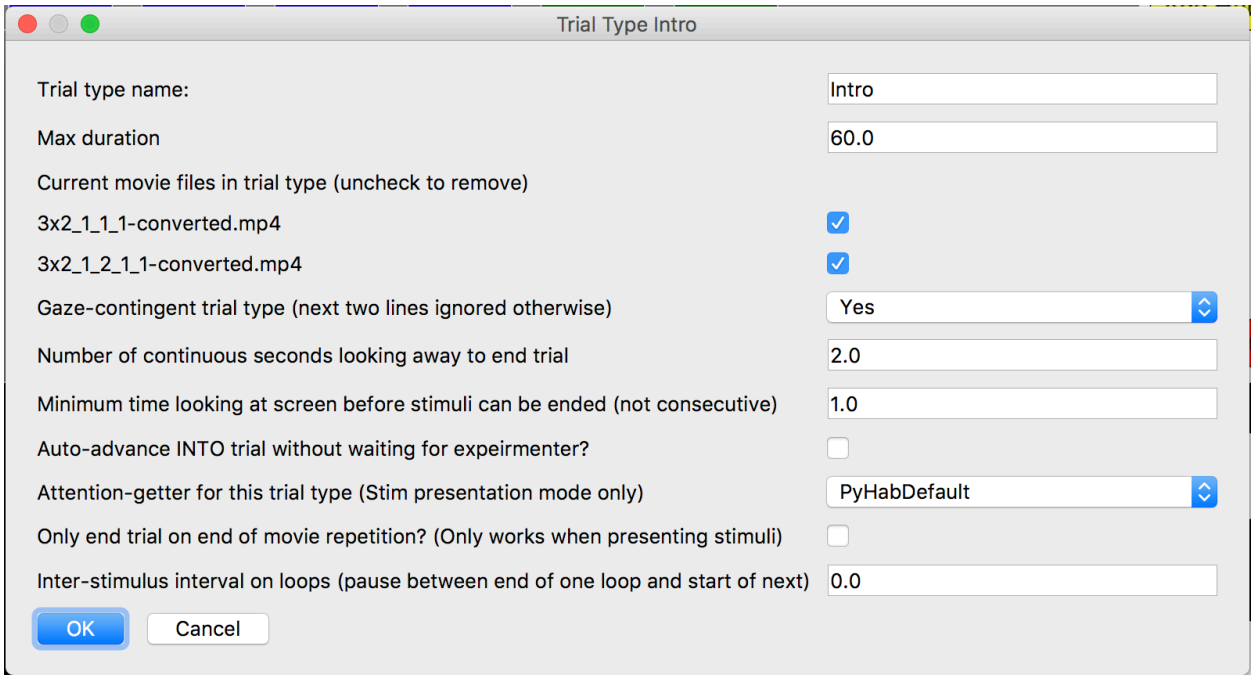


The first line lets you select which trial type you want to assign stimuli to (or if you want to add images to the start and end of the experiment, see below). The second line is how many stimuli you want to add to that trial type. Let's say we want to add two movies to the "Intro" trial type. So, we select "Intro", put 2 in the second box, and hit "OK". Then, you will see this window:



This will have as many drop-down lists as the number you put in the second line of the previous window. **Order matters!** Say we have two intro trials. By default, unless we set up conditions to change it, the first intro trial will play whatever you select for the first line, and the second will play whatever you select for the second line.

If you add more stimuli to the trial type later, it will simply be added on to the end of the list. If you want to remove stimuli from a trial type, you need to go into that trial type's own settings to do it (right-click on the trial type in the trial type palette). This will bring up the trial type window, but with some additional pieces:



Note the two check-boxes after "Max duration". These are all of the stimuli assigned to this trial type. If you want to remove a stimulus from the trial type (but **not** from the experiment's stimulus library), uncheck the box next to its name. Again, **order matters**. The stimuli in a trial type will always appear in the order in which they will be presented by default. If you remove a stimulus from the trial type, everything below it will move up in that order. This will also scramble your conditions a little bit, and you may need to re-make them.

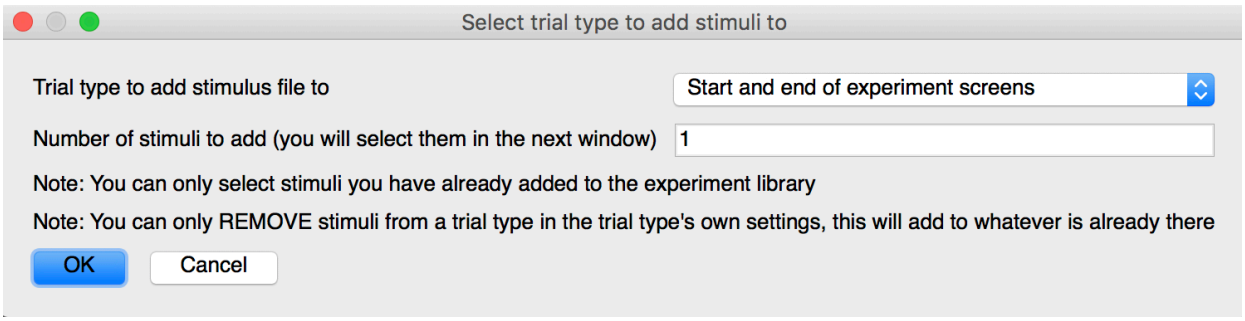
Adding images at the start and end of the experiment

The other thing you can do in this window is add images to the start and end of the experiment. These are completely optional, if you don't add them then the stimulus screen will just be blank (and whatever background color you set in the stimuli settings).

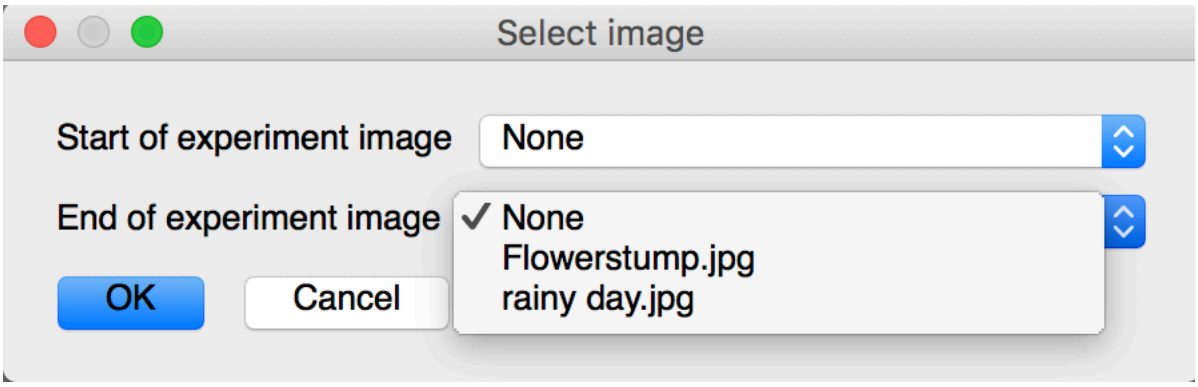
The start image appears when you first start the experiment as soon as the stimulus window opens, and it stays on the screen until you play the attention-getter for the first trial. This is handy if you want to put something on the screen just to get the infant to look at the screen before you start the experiment.

The end image appears when you finish the experiment, while data are being saved, and will stay on the screen until the experimenter presses “return” to close the windows (see ‘Running a PyHab study’). This is useful if, for example, you are using PyHab to present a familiarization phase and then have a test phase that involves some kind of reaching task or other physical interaction, and you don’t want the infant being distracted by your desktop background, or want to have something specific on the screen while they are doing that activity.

In the “assign stimuli to trial types” dialog, at the end of the list of trial types will be “Start and end of experiment screens”. Select this and hit OK. The number field does nothing in this case.



This will open up a new dialog that has two drop-down lists, one for the start-of-experiment image and one for the end-of-experiment image. These will only list the **image files** you have added to the experiment (you can’t use movies or audio for this), and “None”. If you select “None”, the start and end screens will be blank.



Select the image(s) you want to use and hit “OK”.

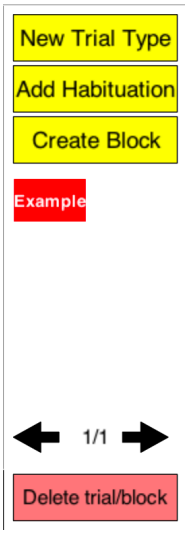
If you have previously assigned a start or end image but want to change it to blank, simply get back to this window and set it to “None” again.

Blocks

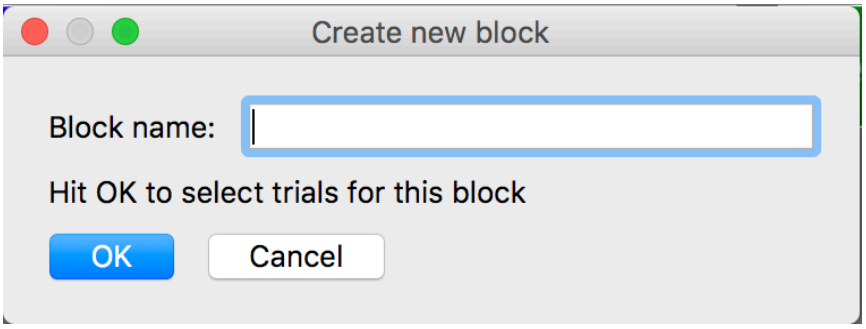
One of PyHab's less intuitive features is that **each instance of a trial type in the study flow will only display exactly one stimulus**. For example, say that you wanted an experiment where you showed a movie that wasn't gaze-contingent, and then showed the last frame of the movie until the infant looked away. One trial could show the movie, but you would need a second trial to show the image. You could make the trial type with the image auto-advance, so that the transition from the movie to the still image was seamless, but you would still need both trials every time you wanted to show the movie.

If you don't want to manually add both trial types every time, you can create a **block**. A block is a collection of multiple trials in one compact package. It adds some powerful capabilities to your study design, but let's start with the basics.

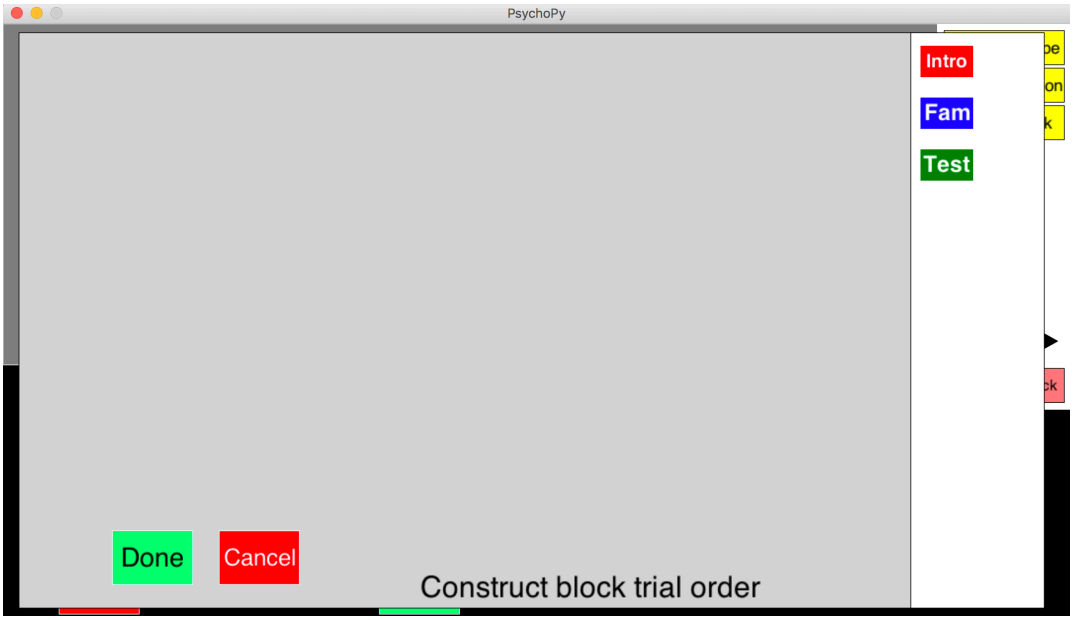
To make a block, first, create the trial types that will go into it. Then, click the 'Create block" button in the trial type palette.



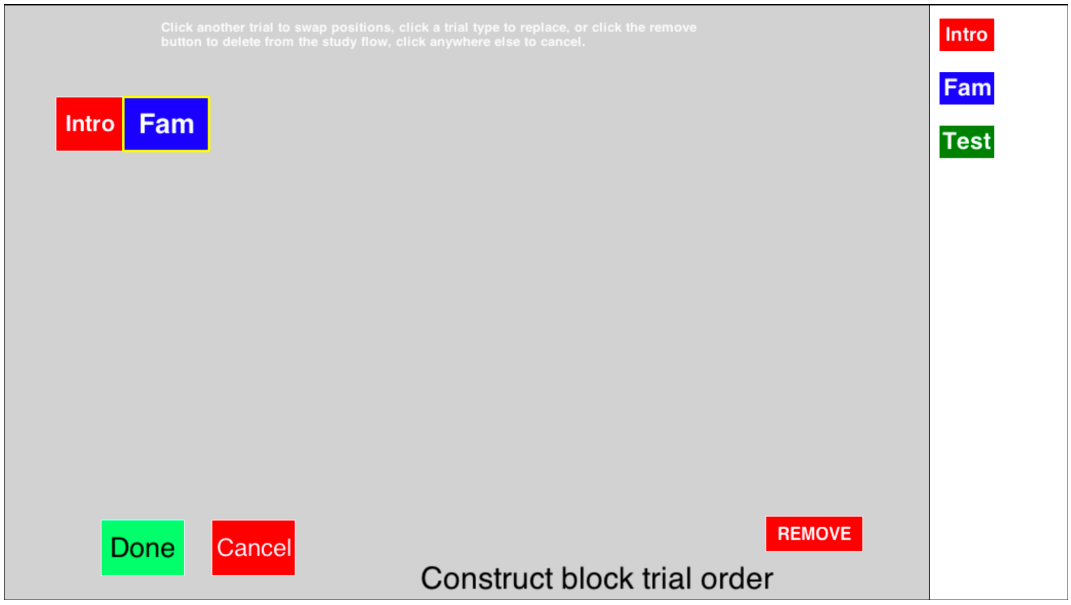
This will open up a dialog that asks you to name the block. This is just like naming a trial type: It's what will show up in the study flow, and in the data files (with a twist - see below). The same restrictions that apply to trial type names also apply to block names: They must be unique, they cannot be 'Hab', and certain characters are excluded. You will get a warning dialog if the block name you provide is invalid.



When you hit “OK”, a new interface will open that allows you to construct the block flow.

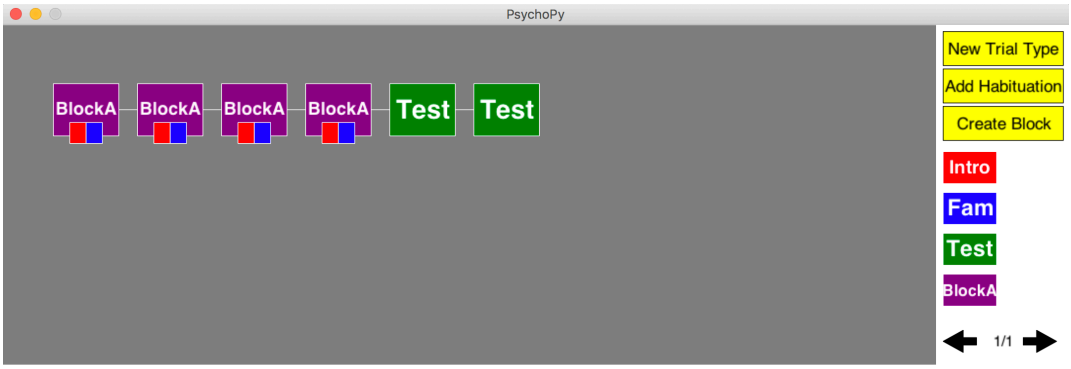


This is exactly like constructing the study flow, except that you can’t modify trial or block types. Click a trial type in the palette to add it to the block flow, click an entry in the flow to switch it with another entry in the flow, swap it with a different trial type from the palette, or remove it from the flow.



When you have finished constructing your block, click “Done”. The new block will appear in the trial type palette, like any other trial type.

Adding a block to the study flow is just like adding a trial to the study flow, but blocks have a little extra information with them:



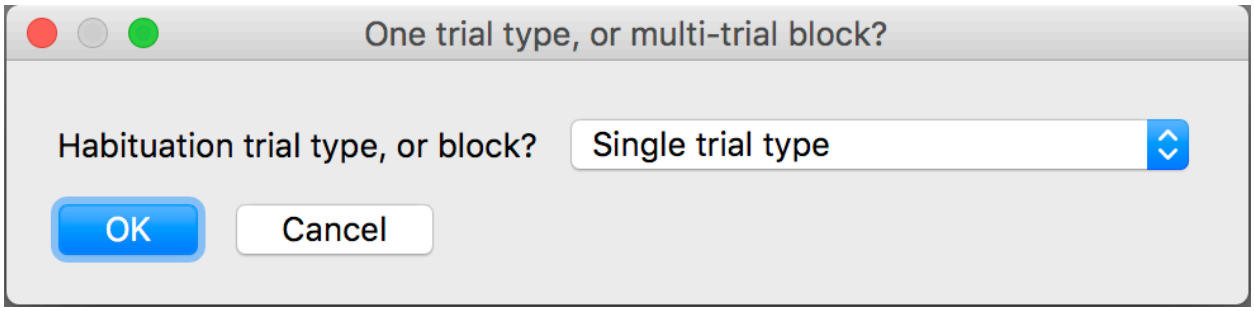
The little squares on the bottom the block's entry in the study flow correspond to the colors of the trial types that are in that block, and their default order. Blocks won't mark whether the first trial in them is an auto-advancing trial or not, but otherwise behave exactly like trial types in the study flow. When you right-click a block in the trial type palette, you can rename it or modify its internal flow.

That covers the basics, but blocks have a few other capabilities as well:

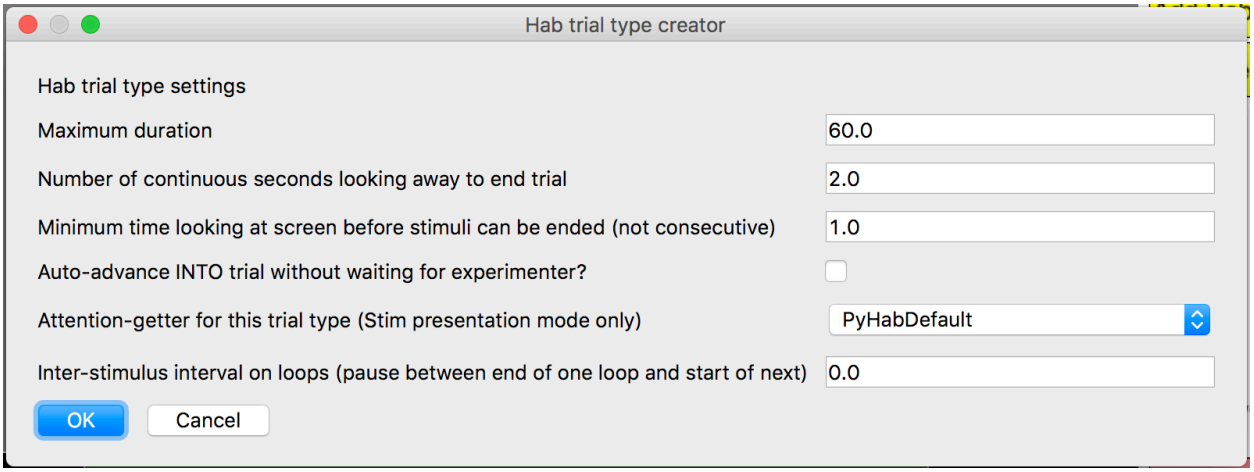
- **Blocks are (almost) recursive**. You can make a block, and then put that block in another block's flow. There are a few restrictions. First, habituation blocks cannot be part of other blocks (though other blocks can be part of habituation blocks). Second, no block can contain itself, or contain a block that contains itself. This is to stop you from accidentally building a study that cannot end. When you are building a block, any illegal options simply won't be available in the trial type palette.
- **Block order can be varied between subjects using conditions**. Say you have two blocks, block X with trials ABCD, and block Y with trials EFGH. Between-subjects, you want to vary both the order of stimuli in the individual trials, but also the order of trials within each block. Using conditions, you can do exactly that. So, one participant could se CBDA and GFEH, while another saw ABCD EFGH.
- **You can get a data summary file that condenses each block into a single line**. Typically, PyHab saves a summary data file with one line per *trial*. In other words, it will expand any blocks to their full extent. However, there is an option to create an additional summary file that condenses all blocks of a given type into a single line. See data settings for more detail.

Habituation

Habituation trials and blocks are a little different from normal trial types. Clicking the "Add Habituation" button will allow you to do one of two things. When you click "Add Habituation", this window will open, and you can select whether you want to make a 'Hab' trial type, or a 'Hab' multi-trial block:

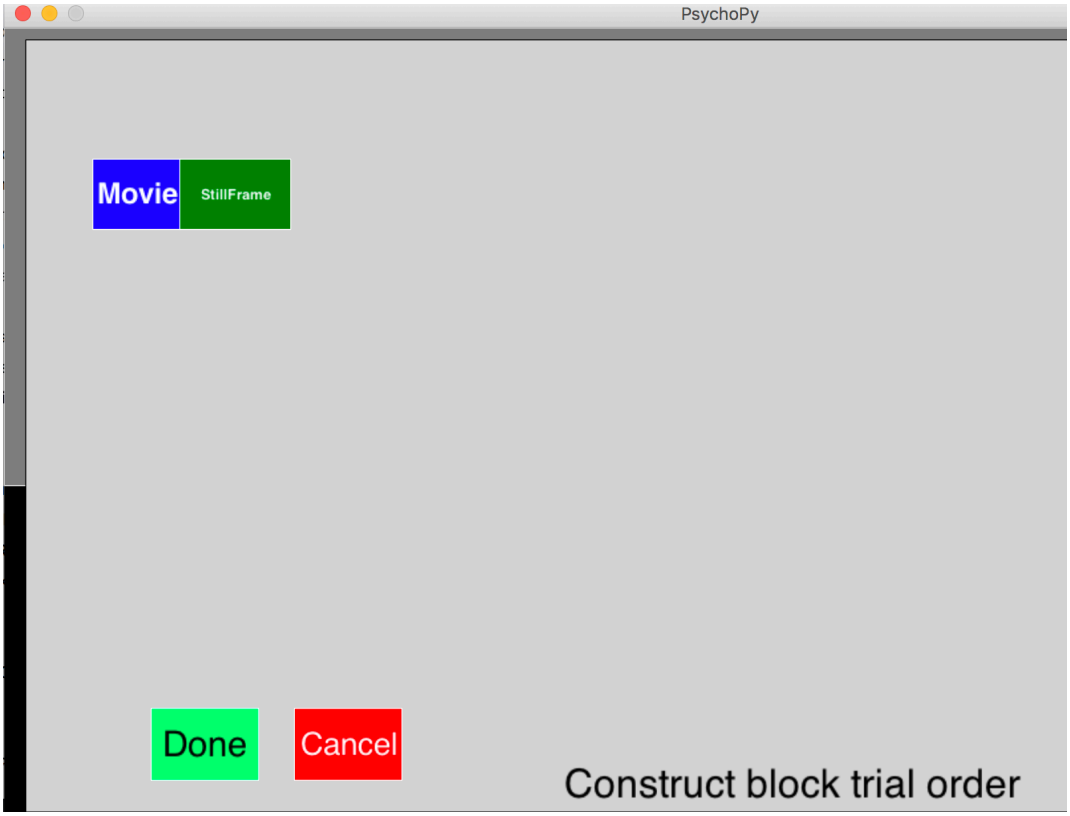


The 'Hab' trial type is a single trial type that will play as many times as specified in the habituation settings. It's a trial that is in most ways like any other trial type, but with certain restrictions. If you just have a simple habituation design where you present a single habituation trial over and over again, that's all you need. Habituation criteria will be computed based on the gaze data for the 'Hab' trial type alone, and all 'Hab' trials will occur consecutively. When you select this option, the following window will open and allow you to modify the properties of the Hab trial like most other trials (except for its name):

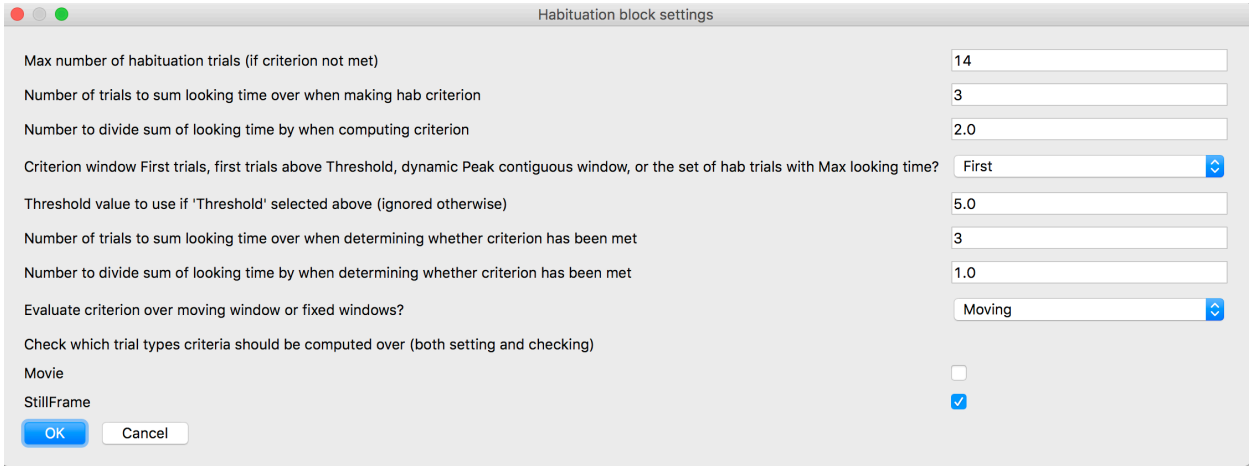


Alternately, you can instead create ‘Hab’ as a block that consists of multiple distinct trial types. ’Hab’ blocks are mostly like other blocks, and you build them using the same interface (see Blocks). The only major difference is that the **habituation criteria, both setting the criteria and determining when they have been met, can be set to look at any of the trials within the ‘Hab’ block**.

Let’s say you had a habituation design where you wanted to show infants a movie followed by a still image of the last frame, but only compute habituation criteria based on how long they looked at the still frame, not how long they looked at the movie. First, you would create a ‘Hab’ block that consisted of the movie trial and the still image.

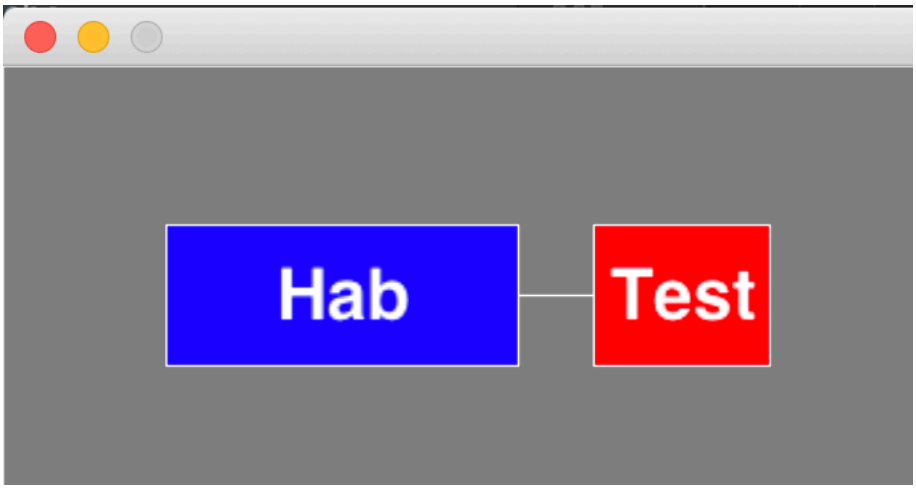


When you hit “Done”, the habituation settings will open, and you will be able to select which of those trials are part of the habituation calculation. So, you would make sure the still-image trial is checked, while the movie trial is not.

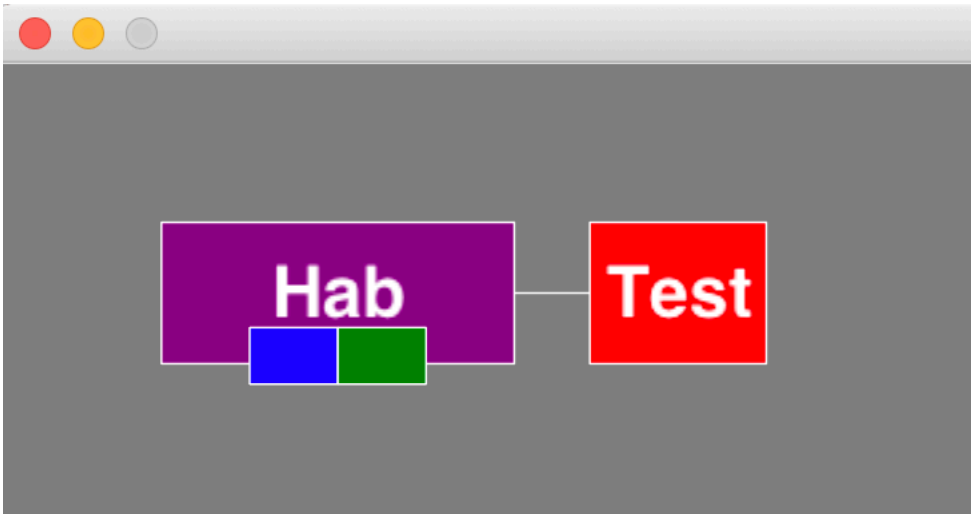


If your experiment contains a 'Hab' *block*, PyHab will automatically save an additional summary data file. Much like the block data file, this condenses each iteration of the 'Hab' block down to a single line. However, unlike the block summary data file, **only data from the trials in the 'Hab' block that are considered in the habituation criteria will be included in the habituation summary data file** (along with any trials that aren't part of the 'Hab' block). So, in the example above, the habituation summary file would not contain any information about gaze behavior during the movie trials, only during the still image trials. The regular summary data file will still be saved with data from each individual trial, so you'll have access to information from these trials, they just won't be part of the habituation summary file.

When you've created your Hab trial or block, it will appear in the trial type palette like any other block or trial type. When you add it to the study flow, it will look a little different:



Hab trial



Hab block

'Hab' shows up double-wide in the study flowbecause it's not just one instance of the habituation trial or block, it's a variable-length habituation sequence, up to the maximum number of habituation trial/block repetitions in the habituation settings, but it will stop earlier if the habituation criteria are reached. So, the study flows above represent up to 14 repetitions (the default setting) of the 'Hab' trial/block, followed by a test trial.

To change from trial to block or vice versa, simply click 'Mod Habituation'.

Deleting trial and block types

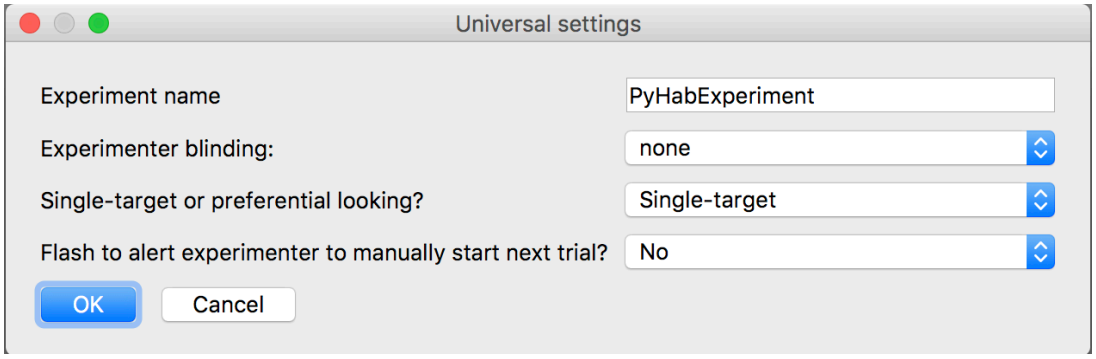
The "delete a trial type" button at the bottom of the trial palette is straightforward. Click it, and a dialog will open with a drop-down list of the trial and block types you have created. Select the one you wish to delete and hit "OK". If you select a trial type, this will remove *everything* associated with that trial type. Any instances of that trial type will be erased from the study flow and any blocks that it appears in, all stimuli associated with it will remain in the experiment library, but not assigned to any trial, and any settings that were specific to that trial type (e.g., maximum duration, ISI) will be forgotten.

If you delete a block, it **will not** delete the trial types in that block, but it will remove any instances of the block from the study flow or from other blocks.

In either case, **you will need to remake any conditions you have made** prior to deleting the trial type, or the experiment will crash when you try to run it.

Settings windows

Universal settings



The “Universal Settings” button will bring up the universal settings dialog, which allows you to control various aspects of the experiment that apply regardless of anything else. Whether or not you are using PyHab for stimulus presentation, no matter what kinds of trials you have, these settings apply to everything you do!

The “experiment name” is the prefix that will be associated with the launcher script for your experiment (see “saving and sharing your experiment”) as well as the prefix for all data files.

There is a separate setting for short delays at the start of a trial, after the attention-getter, but this is “how long after the end of a trial do you want to wait before the experimenter can start the attention-getter for the next trial?” The number is in seconds. Decimals are allowed. That can be found in the Stimuli Settings.

The experimenter blinding settings determine how much information is available to the experimenter while running the study (see Ch. 4). There are three settings:

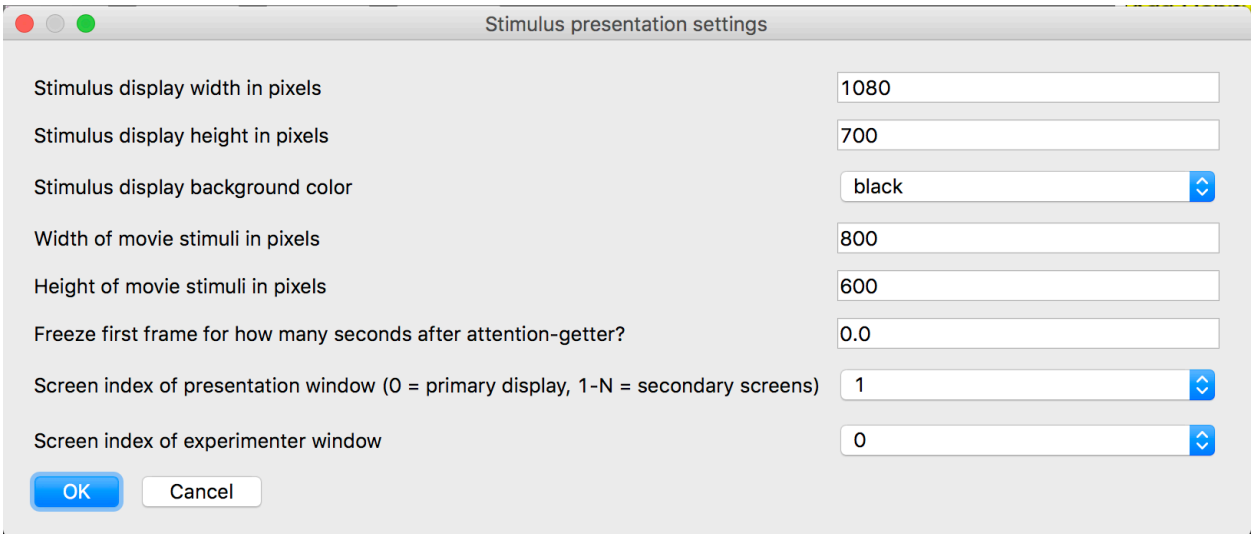
- ‘none’: Maximum information is presented to the experimenter, including trial number, trial type, upcoming trial type, and when each coder is indicating that the infant is looking at the screen or not.

- ‘do not display next trial type’: Experimenter is blind to trial type, but can see trial number and looking coding on/off for each coder. Good for standard blinding to condition/trial type.
- ‘show trial active/not active only’: Maximum blinding. The only information the experimenter sees is whether the trial is active or not. If it is, the coder display box(es) will be blue, and when a trial is not active, they will be black. Great for doing reliability coding because you can’t see the other coder’s coding.

The next setting may be the most important of all: Whether your experiment is single-target or preferential looking. A single-target experiment is one in which you only care whether the participant is looking at the display or not. Preferential looking is when you want to know not just *whether* they are looking at the display but *which* part of the display they are looking at. Preferential looking enables PyHab to code gaze-on L and R, whereas single-target only codes gaze-on and gaze-off. If you have *any* preferential-looking trials in your experiment, this must be set to preferential looking.

Finally, the last setting is for one of PyHab’s optional features. In some setups it’s hard to see the coder window and the view of the infant of the same time, so it can be difficult to know when a trial has ended and experimenter input is required to start up the next trial. This setting allows you to make the experimenter window flash briefly to let the experimenter know the next trial must be manually started. The participant doesn’t see anything and there are no audio cues, it’s just to make it a little more obvious when the experiment is between trials.

Stimuli settings



Clicking the "stimuli settings" button will bring up this dialog. If you are not using PyHab to control your stimulus presentation, the last three lines are potentially relevant, but nothing else is. If you *are* using PyHab to control stimulus presentation, then all of this matters.

The top two lines are the width and height of the stimulus display area. Generally speaking this will be the resolution of your presentation screen If you're using a 1024x768 resolution, for example, you would put 1024 in the top box and 768 in the second.

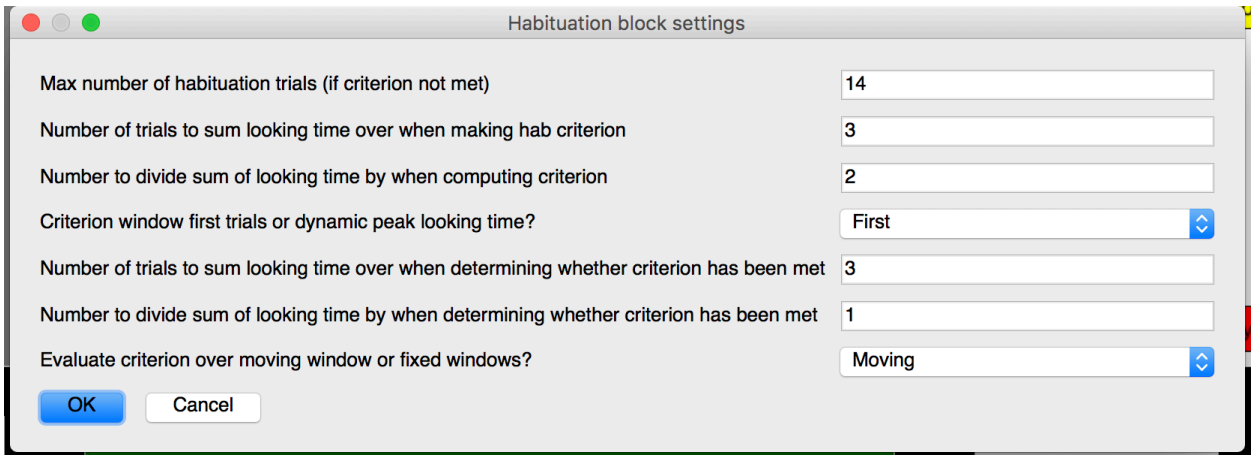
The next line is the background color of the window. By default, it is black, but you can also make it white or gray.

The next two lines are the width and height of the stimuli *within* that display area. If you want your stimuli to fill the entire screen, these numbers should match the first two. If they are not the same, the blank background will be visible behind the stimuli.

The next line imposes a minimum delay between the end of the attention-getter and the start of the stimulus. I recommend setting this to at least .1 if you are using PyHab to control stimulus presentation, because if there is no delay the sound from the attention-getter can conflict with any sound in your stimuli. If you aren't using PyHab for stimulus presentation and auto-advance is not checked for a given trial type, then when you hit the attention-getter key there is a delay of 500ms + whatever is in this box. This is useful for matching the duration of an attention-getter in a recording, for example.

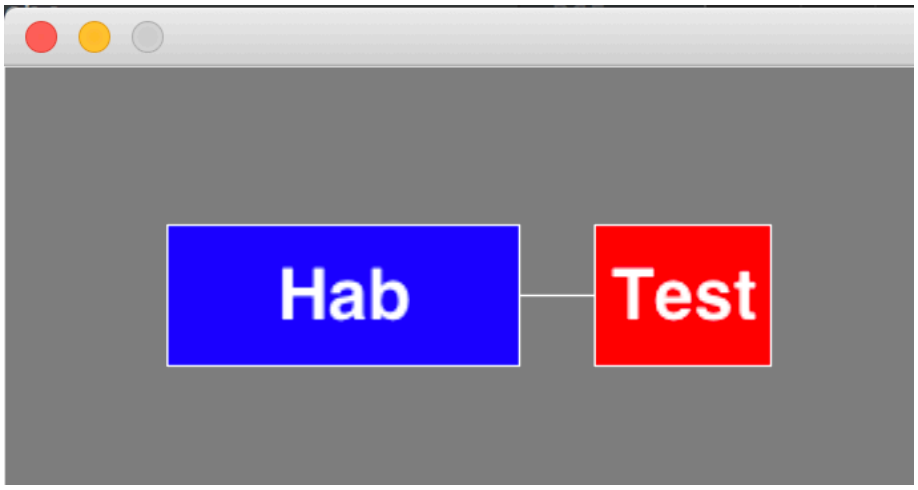
The “Screen index” fields are basically “which monitor should the stimuli appear on, and which monitor should the experimenter appear on”. By default, on most computers, the “secondary monitor” will be screen index 1, while the primary display will be screen index 0. The drop-down menu will show you all of the available screen indexes (if you are building it on a computer with only one screen, it will default to ‘0’ and ‘1’, on the assumption that you will be plugging in an external monitor later).

Habituation settings



If you click the “Habituation settings” button, you will see this dialog. These settings only matter if your experiment includes a habituation sequence, the length of which is determined by the habituation criterion.

The first line is the maximum number of habituation trials, period. Even if the infant never reaches your habituation criterion, after this many habituation trials, another trial type will be shown. For example, in this study flow:



After *at most* 14 habituation trials, the test trial will be shown, no matter what. If you have a 'Hab' *block* rather than a trial, they would see at most 14 repetitions of the block (regardless of the number of trials in the block that were included in the habituation calculations).

> **Note:** 'Hab' blocks will always play every trial in the block on every repetition, and won't run habituation calculations until after the last trial on each loop.

The next three lines determine how the criterion is determined initially. The first number is how many trials you sum the looking time across. The second number is what you divide that total looking time by to set your criterion. So, with the settings pictured, the habituation criterion would be the sum of the looking time across the first three habituation trials, divided by two. The last line determines what trials are examined when setting the criteria. "First" means it will always be calculated from the first Hab trial. "Peak" means that it will check after every Hab trial whether the criterion is greater if it were set over the most recent N trials, where N is the number you put in the first line. If so, it updates the criterion to the greater value. In short, it's setting criteria based on the window of peak looking. "Max" is similar to "Peak", but instead of looking for three *contiguous* Hab trials, it will just take the three Hab trials with the longest looking time, period.

The following three lines determine what looking time is compared to the habituation criterion to determine if the participant has habituated to the stimuli. The first number is again the number of trials to sum looking times across. This picks up from the first Hab trial after the

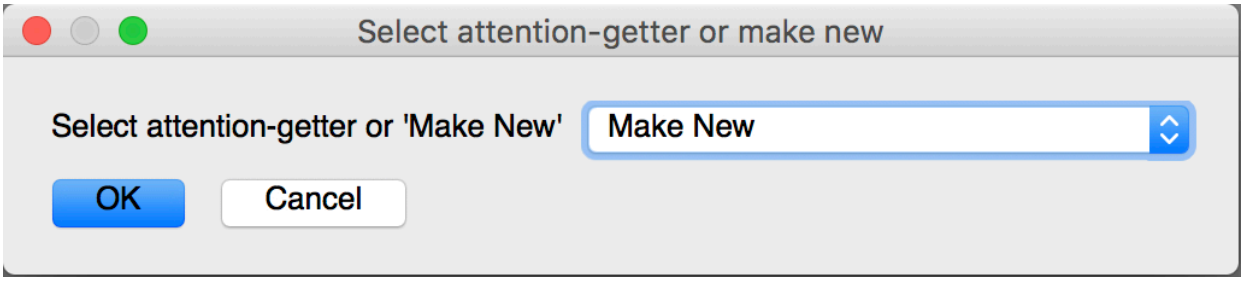
criterion has been set initially. The second number is then what the total looking time in those trials is divided by before comparing it to the criterion. The last line determines whether the criterion is evaluated over a moving window of trials (so the most recent N trials, where N is the number of trials to sum across), or fixed windows (after every N trials once the criterion has been set).

With the settings above, after the 6th habituation trial, the program would compare the sum of looking times across trials 4-6 (divided by 1, so, just the sum) to half the sum of the looking times of trials 1-3, and if that value was less than the criterion, the participant would be considered habituated and the test trial would be shown. If the sum of looking times across those three trials was greater than the criterion, after the next habituation trial the program would sum looking times across trials 5-7, and so on and so forth until the sum is less than the criterion or the maximum number of habituation trials has been reached. If it the criterion were set to "Fixed" rather than moving, it would only check against the criterion again on trial 9, and if it is not met then, trial 12.

Attention-getters

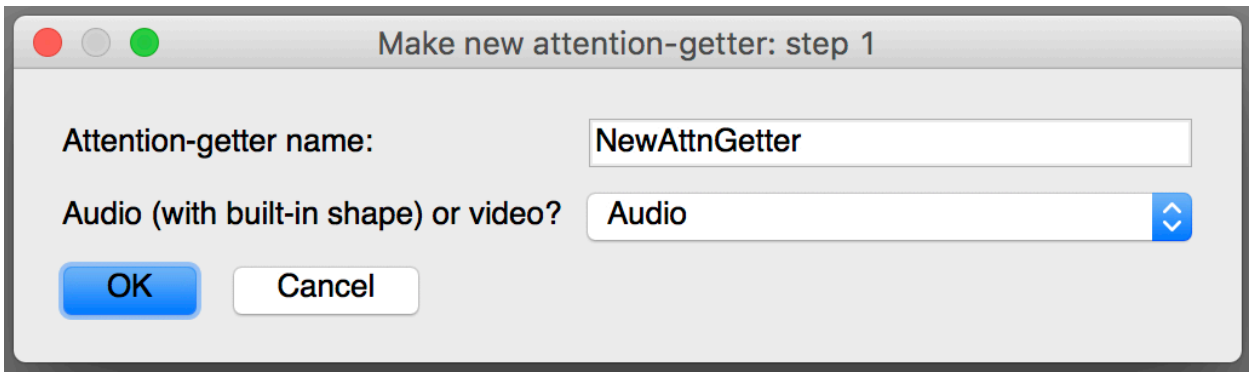
As of version 0.5, PyHab has the ability to let you customize your attention-getters, and use different attention-getters for different trials. There is a default attention-getter which is always available, which consists of a looming yellow rectangle and a rising musical scale. However, you can make your own, either a movie file, or an audio file with a procedurally-built shape.

When you click the "customize attention-getter" button, the first window you will see is this one:



If you have already made some attention-getters, they will be in the list. Otherwise, the list will only have the “Make New” option. The “Make New” option will always be present.

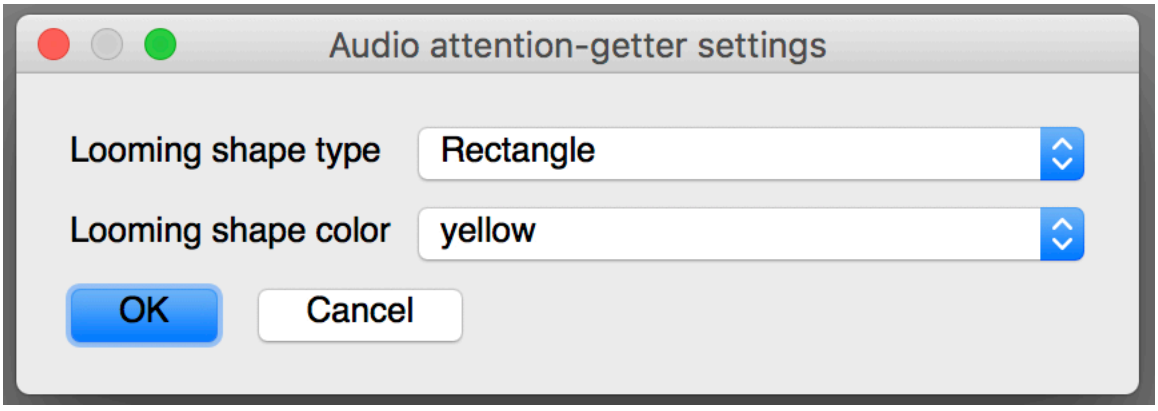
When you hit OK, the next window that pops up will look like this:



The first line lets you name your attention-getter. This name will appear in the trial type dialog’s list of attention-getters. The second line lets you choose what kind of attention-getter you want to make. There are only two, audio and video. Select which one you want, and hit OK.

Video is in some ways the simplest: An open-file dialog will appear, and you select a movie file that you have made to use as your attention-getter. The same movie formats that work for stimuli should work for your attention-getter as well.

For audio, there are two steps. The first step is to customize the shape that goes with the audio. A window will appear that looks like this:



The shape will be rotating and looming. You can choose a rectangle, cross, or star, and one of six colors. When you hit OK, you will then see an open-file dialog and select the audio file you want to use. Again, all the formats that work for stimuli should work for this as well, and for audio that’s most of them. The rotation and looming will be synchronized to the length of the audio file automatically.

The data settings window is just a long list of fields that will appear in your summary data file, with a check-box next to every field. Most field names are self-explanatory, with a few exceptions:

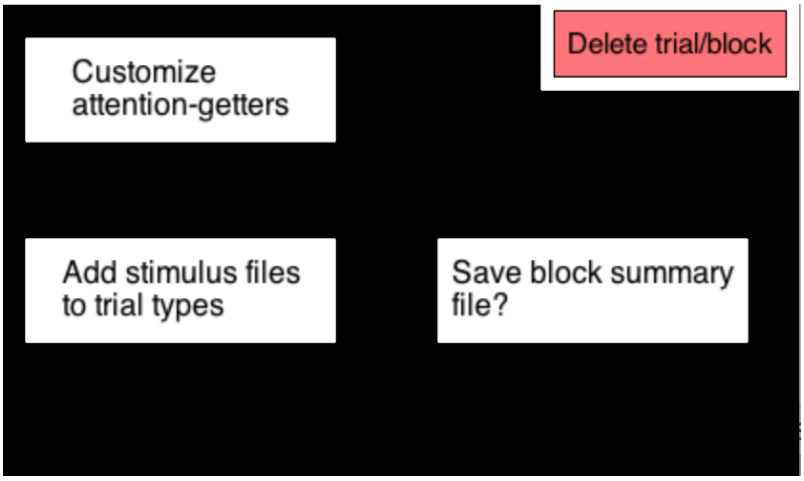
- GNG: Usable/not usable trial. 1 if the trial is usable, 0 if trial was aborted or redone.
- stimName: The name of the movie file (in a stimulus presentation version, blank otherwise)
- condLabel: The label of the condition selected by the experimenter, if condition randomization is used. If you are not using condition randomization, this will simply match the "cond" field.

There are two extremely important things to know about these settings:

1. **Anything not checked will not be recorded in your data in any form**. There will be no hidden logs or anything of the sort, that information will simply not exist.
2. **If you change your study from single-target to preferential looking, these options will reset, check every box (so all data is recorded by default), and change the list for the appropriate set of columns**. The data recorded are actually different depending on what kind of study you are running. The program always defaults to including everything, but if for whatever reason you want to omit a column, you will have to revisit these settings if you switch from single-target to preferential looking.

Block data settings

When you create a block for the first time, an additional option will appear in the bottom-right most area of the main interface that says “Save block summary file?”.

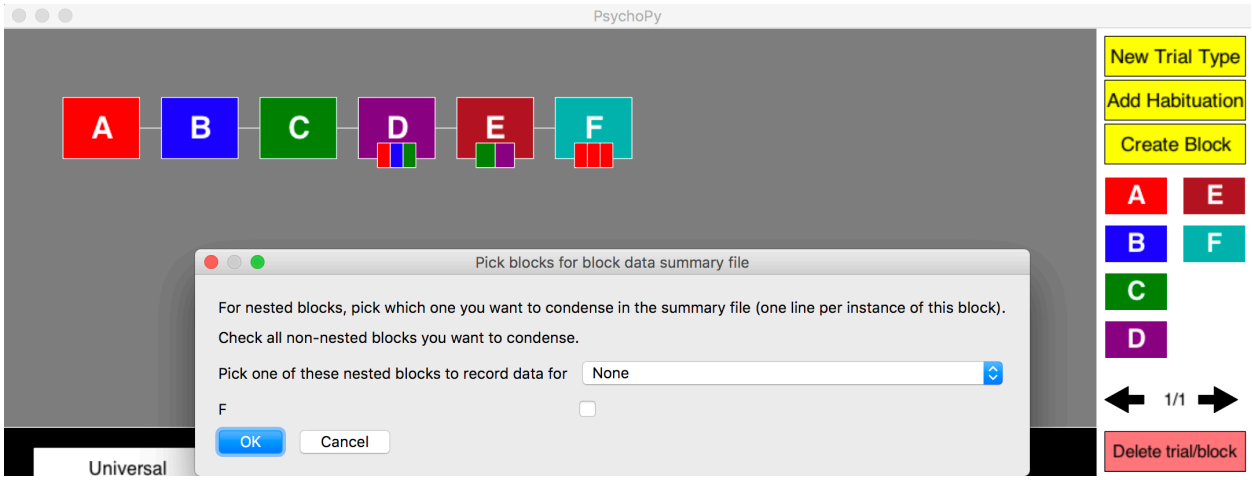


This gives you the ability to save an **additional** summary data file in which all of the trials belonging to a given block are compressed down into one line for each instance of the block. **Note that you will still get the regular summary data file, which includes every single trial in the experiment**. So, for example, if you had a block with four different trials in it and you cared about the total looking across all of them, and you didn't want to have to manually add up four lines from the main summary data file for your analyses, you could use this to make a separate block summary file that did that math for you. Any trials not in the block in question will be recorded as normal.

When you click on this button, you get a simple dialog masking a complex problem. Here's the short version: I didn't want you to be able to save data from the same trial twice. That means that if you create a block that itself contains another block, I only wanted you to be able to summarize **one** of those blocks, because if you summarized both, data from the 'lower-level' block would be repeated in the lines for the 'higher-level' block.

My solution to this was: For blocks that are not "nested", that is, do not appear within other blocks and only contain trial types, not other blocks, there is a simple check-box. If the box is checked, the block will be summarized in the block data file. For nested blocks, every nested block will appear in a single drop-down menu, from which you can select one to be summarized. If you summarize a lower-level block, other trials in the upper-level blocks will appear as their own lines as normal. If you summarize an upper-level block, it will condense **all** of the trials from all of the lower-level blocks into one line.

Let's take an example of how this setting window would look in an unusual study design:



From the study flow, you can see that A, B, and C are all simple trial types. D is a block that consists of A, B, and C. E is a block that consist of trial type C and block D. F is a block that consist of three repetitions of A.

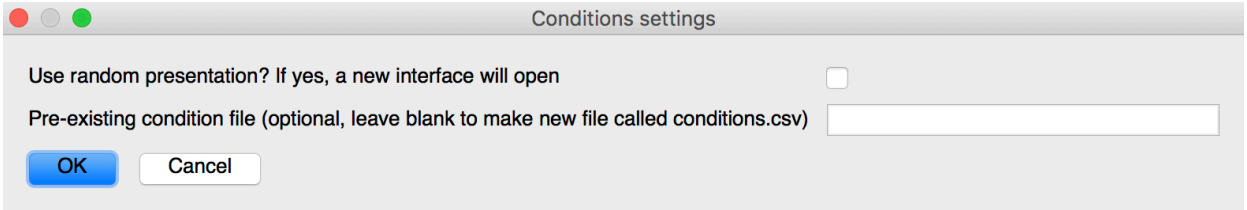
So, D is 'nested' within E. When you open up the block summary data settings, there is a drop-down list where you can select "D", "E", or "None", and the block summary file will summarize whichever one you select. If you select "D", then the block summary data file will include, for every instance of E in the study, a line for C and one line for all of block D. If you select "E", then the block summary data file will include one line for every instance of E. F, because it is not nested in any other block and contains only basic trial types, appears as a separate check-box, and checking it will mean that every instance of F gets only one line in the block summary data file.

Here is where things get a little complicated. Before you start messing with these settings, here's what you need to know:

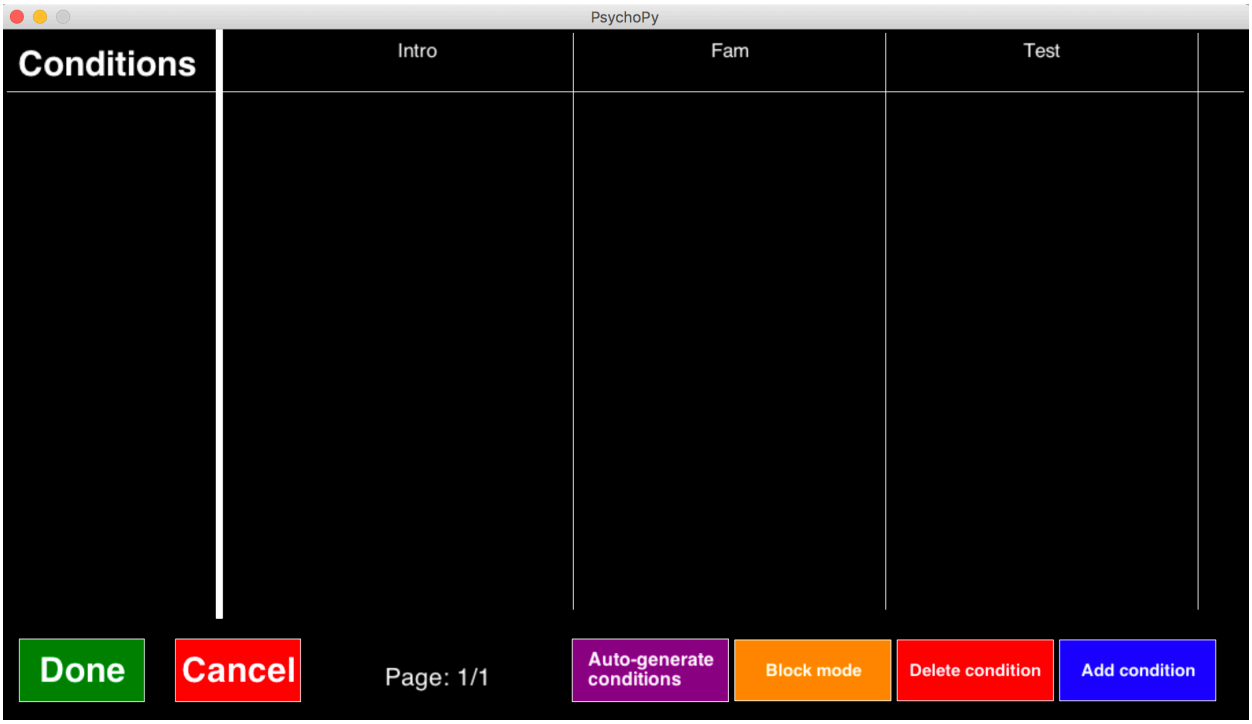
1. If you are using PyHab for stimulus presentation, you *must* have all of your movie files attached to all of your trial types before setting up conditions. If you do not, or if you change the movie files later, you will need to redo the condition settings.
2. If you are not using PyHab for stimulus presentation, you will be given the option to manually enter a list of conditions so that you can use a drop-down menu when running the study, instead of entering the condition manually. This list must be enclosed in square

brackets, each condition label enclosed in single quotes, separated by commas. For example: [‘Condition A’, ‘Condition B’, ‘Condition C’]

I will focus on condition settings when you are using PyHab for stimulus presentation, because that’s when it actually matters. Once you have all of your movies associated with their trial types, click on “condition settings” and you will see this dialog:



If you want to use condition randomization, check that box. If you already have a condition file (and you can create them manually), you can enter the filename in the second line. If you have movie files associated with trial types, check the box, and hit OK, the whole builder interface will change to the condition interface, which looks like this:



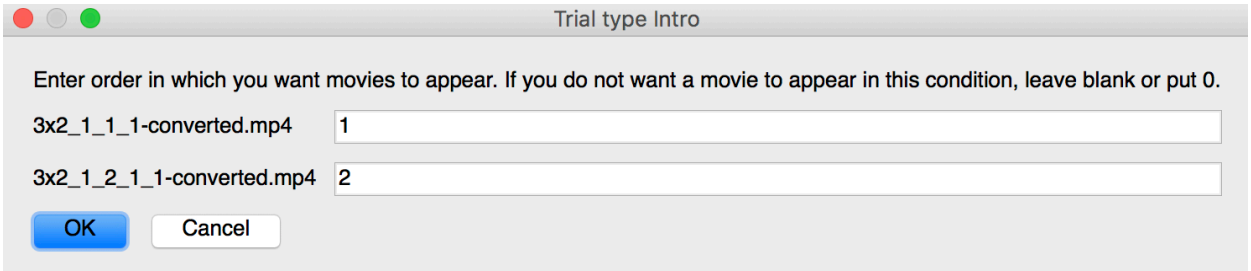
The “Done” button in the bottom-left saves your conditions and exits to the regular builder. The “Cancel” button just exits to the builder but doesn’t save anything. Across the top you have the list of conditions in the left column, and then each trial type in another column. There are

four other buttons across the bottom. From the right, “Add condition” and “Delete condition” let you manually create or remove a condition. “Block/Trial mode” switches between a view that shows you the order of stimuli within trial types, or the order of trial types within blocks (you can change both, but only one at a time). The last button will say either “Auto-generate conditions” (if there are no conditions present) or “Randomize over subjects” (if there are conditions already). The condition auto-generation system allows you to automatically generate permutations of the orders of your stimuli or trials within blocks without having to specify each one manually. “Randomize over subjects” allows you to create a blinded, counterbalanced random order of conditions.

Each function is explained below.

Add condition

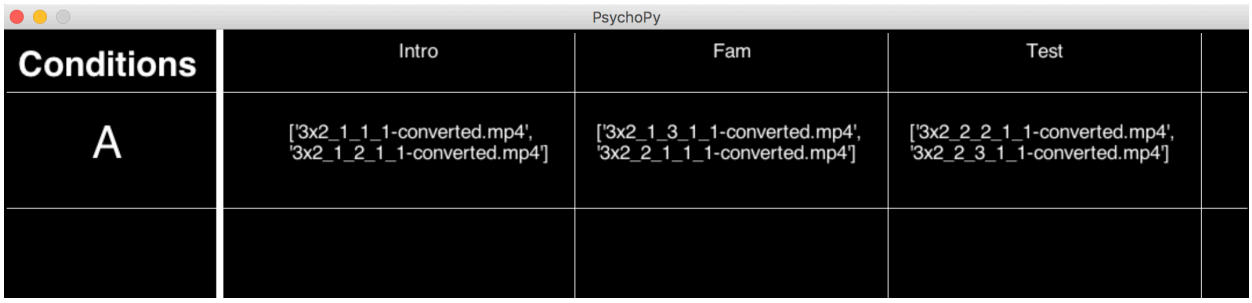
When you click the “Add condition” button, you will go through a series of dialogs. In the first one, you simply set the name of the condition, which will appear in a drop-down menu when you run the study. Then, you will go through a separate dialog for each trial type (according to the order in which the trial types were created, not their order in the experiment). In each of those dialogs, you will see a list of all the movies in that trial type, with numbers next to them. For example, in the demo experiment, if you go to this menu and hit “add condition”, after naming the condition you will see this dialog for the intro trial type:



The numbers represent the order of the movies *in that condition*. So, if in this particular condition you wanted the second movie to appear first, you would simply switch these two numbers. Remember earlier when I mentioned that when you have multiple instances of a trial type in your study flow, it goes through the movies associated with that trial type in order, and loops around as needed? Here is where you determine that order, if you want it to be something other than the default.

In addition, if you want to have a between-subjects manipulation such that different participants see entirely different sets of movies, you can do that with this interface as well. **If you do not want a movie to be presented in a given condition, simply leave the field next to it blank or put a 0. For participants in that condition, it will be as if that movie does not exist.**

When you hit "OK", you will then be prompted to do this again for each of the remaining trial types. After the last trial type, you will be returned to the condition screen, and the condition will now be listed.



The list of stimulus names will appear in each cell, in the order that they will be presented in that condition.

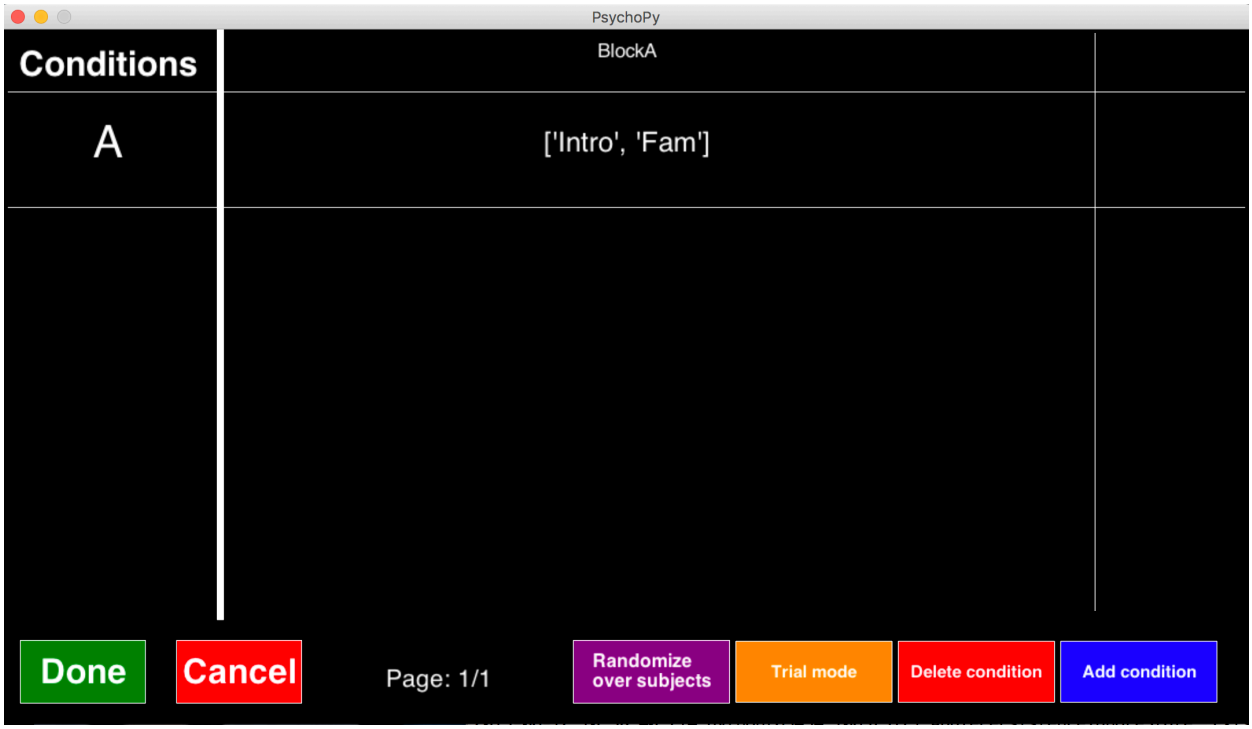
You can modify a condition by clicking on the row it occupies, and delete it using the "delete condition" button, which will bring up a list of conditions much like the "delete trial type" button. You can have as many conditions as you want.

If you have more than four conditions, the conditions will be spread over multiple pages, and next/last page buttons (arrows) will appear on either side of the page counter on the bottom. Click the down arrow to go to the next page, and the up arrow to go to the previous page.

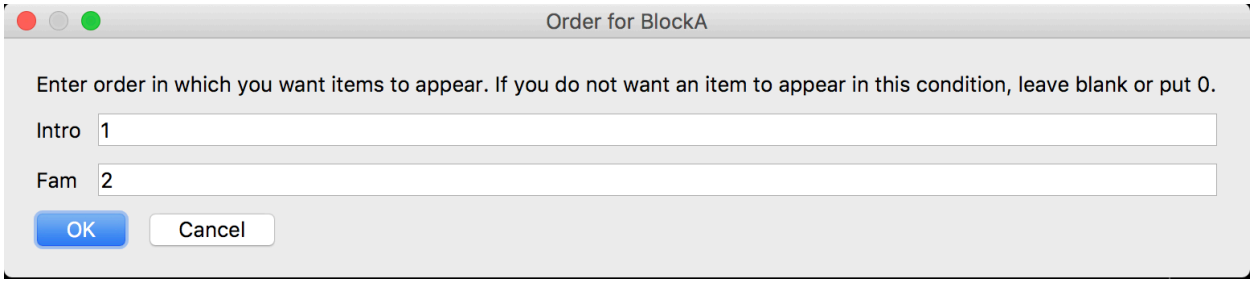
When you are finished simply click "Done" to be returned to the regular builder screen.

Block/Trial mode

Block mode works exactly like trial mode, except that instead of letting you change the order of stimuli within a trial type, you can change the order of trial types within a block. If you clicked “Block mode” after making the condition described above, you would see this:



If you are in block mode and click “Add condition”, or click on an existing condition, you will get a series of dialogs just like you do in trial mode, except for the trials within each block instead of the stimuli within each trial.



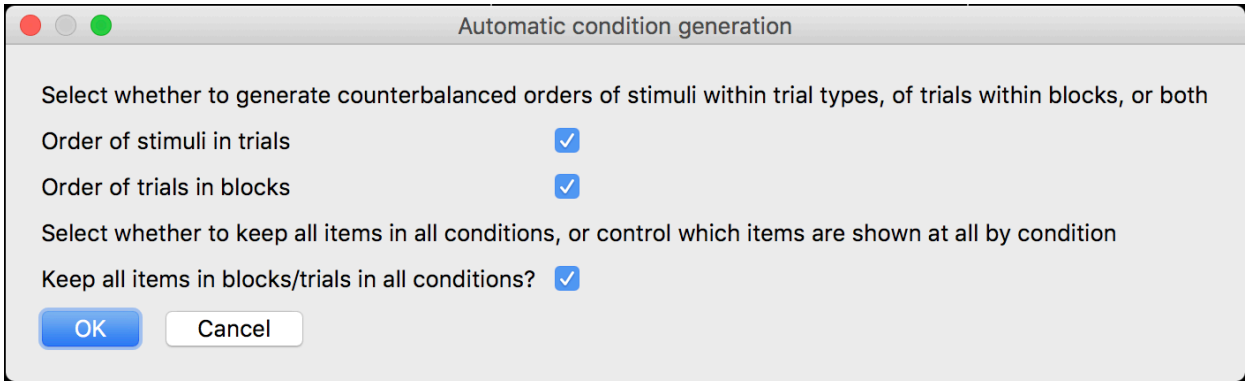
The same rules apply: Consecutive ordering, 0 or blank to omit a trial type from a block altogether in that condition.

Note that, when creating a new condition, if you want to vary both the stimuli in the trials and the trials within the block, you will need to create the condition in one mode,

switch modes, and then modify it in the other. Any block or trial not specified will be left in its default order.

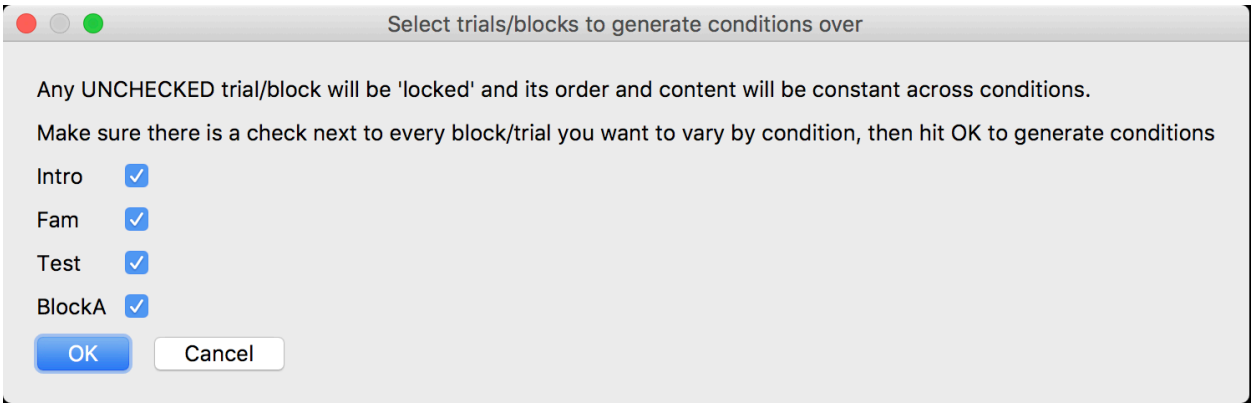
Auto-generate conditions

The automatic condition generation system was requested by PyHab users who got tired of manually creating dozens of different conditions for their more complex study designs. In short, it asks you what you want to permute and how, and based on that information, creates conditions covering every possible permutation of orders for the trials or blocks you specify. When you click "Auto-generate conditions", the first dialog that opens will look like this (if you have both blocks and trials, otherwise only the last line will be present):

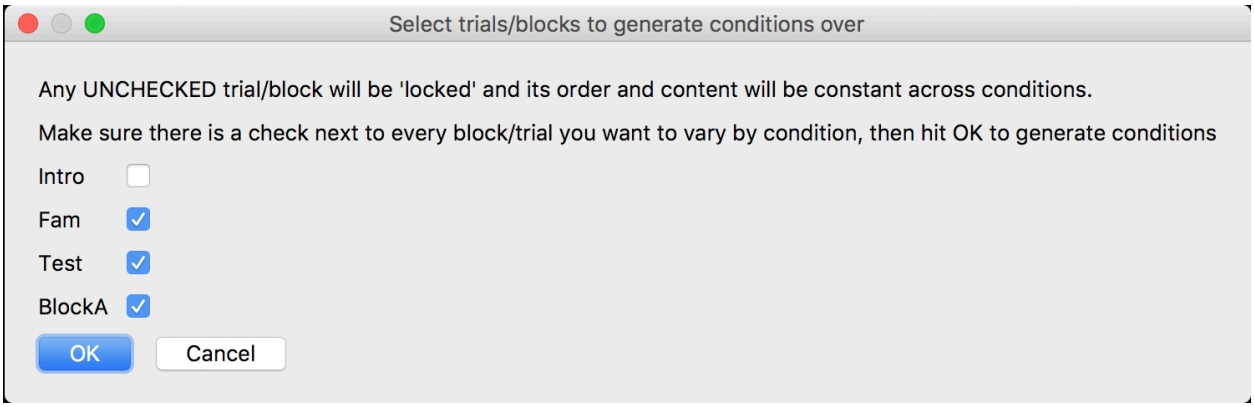


You can choose to generate conditions that counter-balance the order of stimuli in trials, the order of trials in blocks, or both. The last check-box controls whether the generated conditions ensure that every participant sees every trial and stimulus, or if they manipulate, between-subjects, which things appear at all. If the last box is unchecked, you will have the ability to design a between-subjects manipulation, and there will be an additional dialog box (see below).

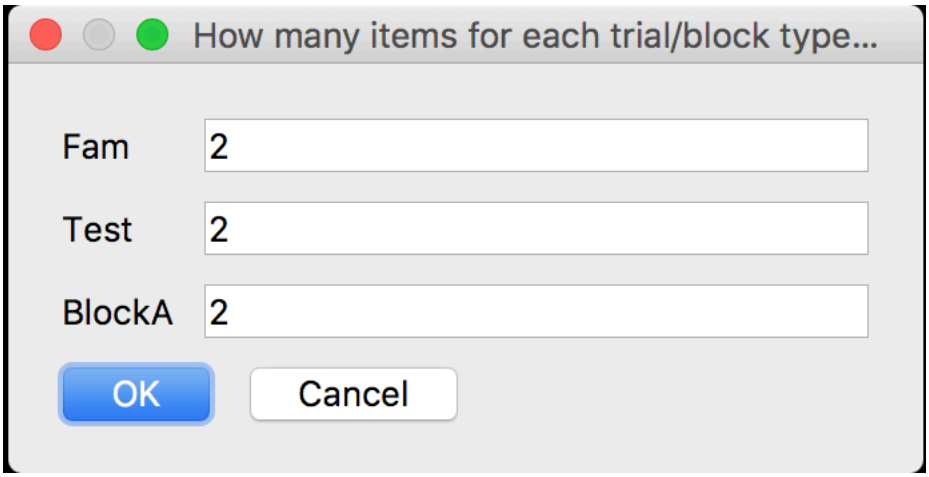
When you click "OK", the next dialog that appears will ask you if you want to "lock" any trials or blocks.



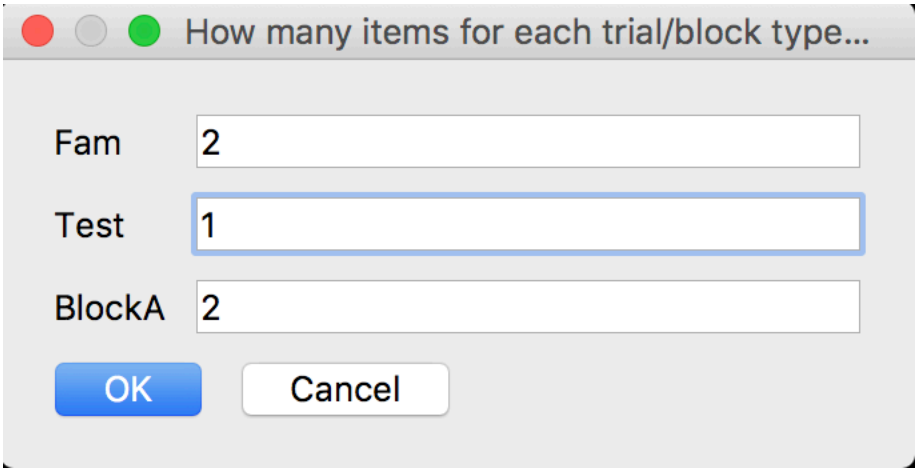
If you don't want to counter-balance the stimuli in a given condition or the order of trials in a given block, simply un-check it here, and items in that category will appear in their default order in every condition. So, for example, if we wanted the intro trials to always be the same, we would uncheck the first box like so:



When you hit "OK" here, **if you unchecked the box in the first dialog that says "keep all items in blocks/trials in all conditions", an additional dialog will appear**. This dialog lets you select how many items will be in each condition for a given trial or block.

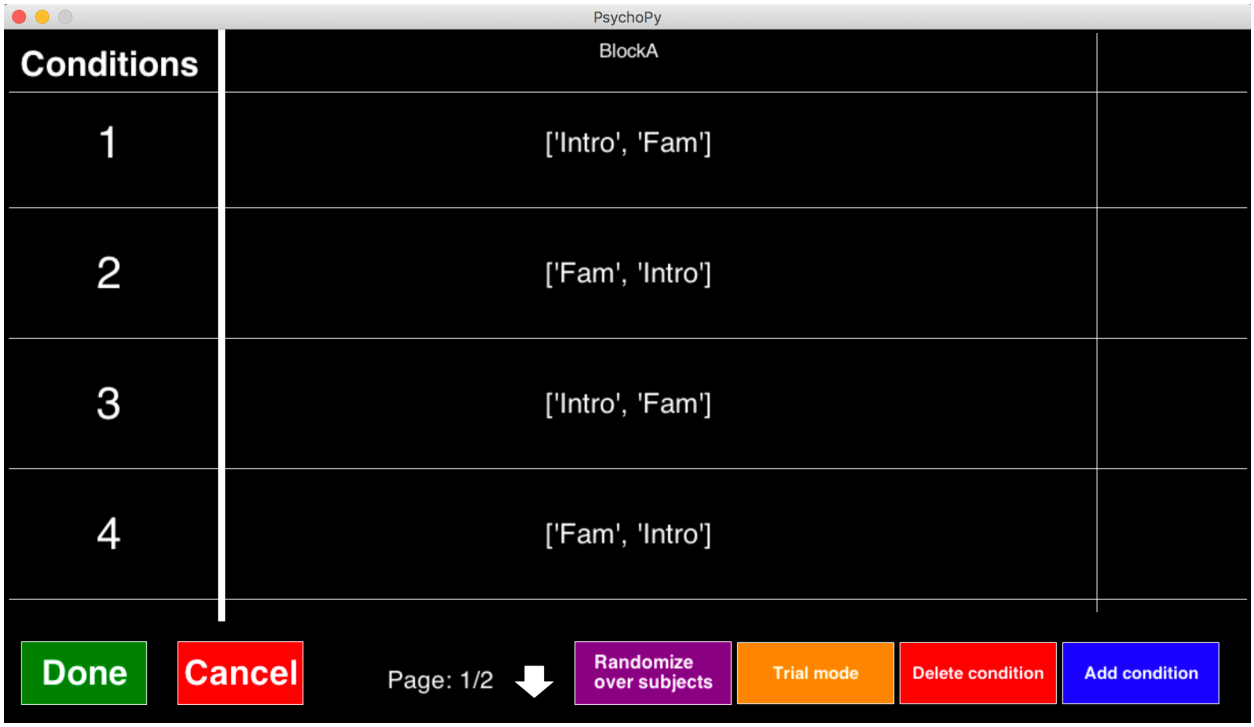


Note that the "locked" trial type, "Intro", doesn't appear here. Now say that we wanted to make it so that each participant only saw **one** stimulus during test trials. So, a between-subjects manipulation. We would simply set "Test" to only have one item per condition:



Finally, when you hit "OK", your conditions will be generated. They will, by default, be numbered.

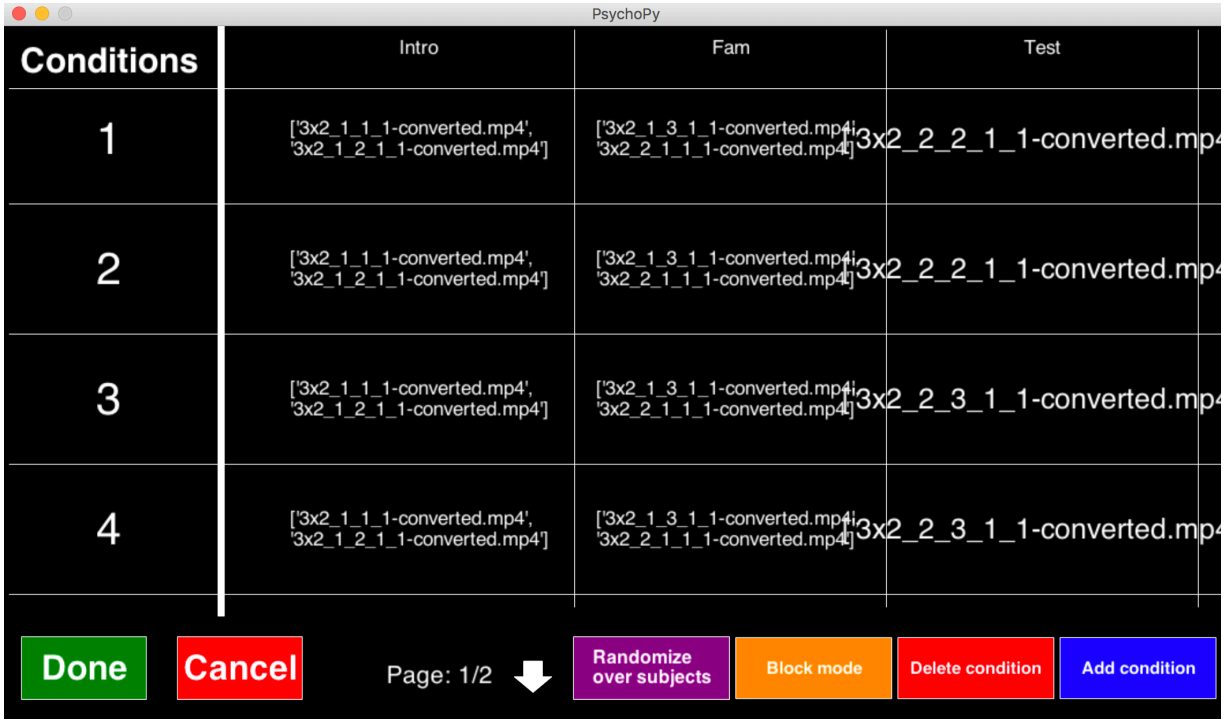
Note that there are a total of 8 different conditions across 2 pages. "But wait", you might be thinking, "aren't conditions 1 and 2 completely identical?" Not quite. If you click on "Block mode":



The order of stimuli in the Fam and Test trials is identical, but the order of trials in BlockA is not. **Every possible permutation is created within the constraints that you specify**. That means you might have to be careful about locking certain blocks or trials when generating conditions. If you didn't get the conditions you wanted, just hit "Cancel", re-open the condition settings, and try again. As long as there are no conditions in the condition list, you can use the automatic condition generation system.

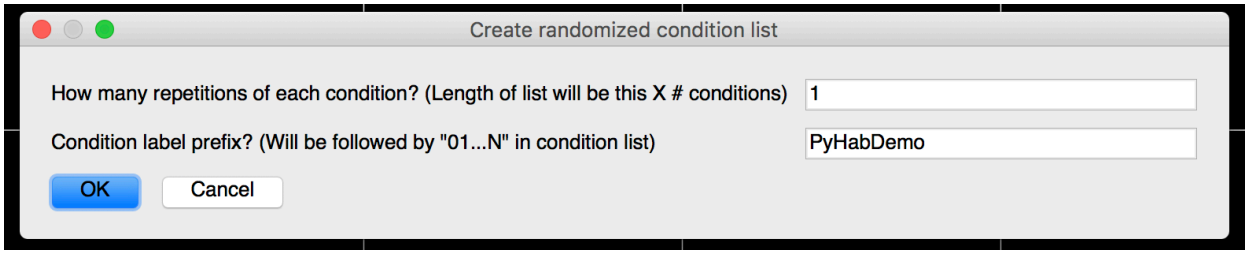
Creating randomized conditions by participant

Let's say you created eight different conditions, like so:



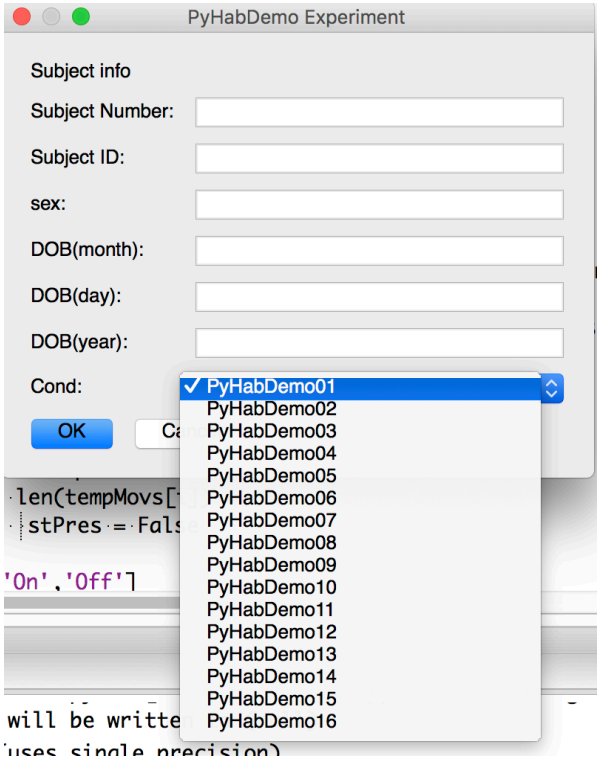
Say you plan to recruit 16 participants for your experiment, and you want to randomly but evenly assign them to each of these eight conditions. Furthermore, you want to be blind to the condition for each individual participant, so the person running the study doesn't know what condition they are currently running. What you ideally want, in that case, is a separate condition label for each participant that maps on to one of these eight conditions.

To make this easy, we have the "Randomize over subjects" button. This button takes the conditions you have **already made**, and creates a condition list with a set number of repetitions of those conditions, and doesn't show you the end result. If you click the purple button, this is what you will see:



The first line is how many times you want each condition to show up in the final condition list. (Currently it can only give each condition the same number of repetitions). So, if you have 16 subjects, that's 2 repetitions of each of your 8 conditions. So, you would put 2 in that box.

The second line is how each of your conditions will be labeled. This is what will appear in the condition list when you run the study, followed by a number. When you are done, hit “OK”, and PyHab will close the condition menu altogether. When you save your PyHab experiment, the new condition file will be saved. When you run the study, the condition dropdown will now look like this:



One condition for each participant, **and the condition order has been randomized.** Two of these correspond to condition 1, two correspond to 2, etc., but you don’t know which one is which. However, the presentation order will appear in the data file as long as the “cond” column is selected in the data settings (and it is by default), so you will always know exactly what each participant saw after you run them. If you want to be blind to which condition each participant is assigned to, you’re done! **If at any point you want to un-blind yourself, simply re-open the condition menu in the builder, or open the condition csv file itself.**

c. Saving and sharing your experiment

When you first create an experiment, there will be a “save as” button and a “quit” button in the bottom left of the builder window. The quit button will also prompt you to save before you quit. When you save a study for the first time, a standard save dialog will open. Choose where you want to save the study to, type a name for the folder the study will occupy, and hit save.

This will create a completely self-contained folder that includes everything needed to build and modify your study. In this folder will be a file named “[studyname]Launcher.py”. Whenever you want to run or modify your study, simply open this script in PsychoPy and run it. It will launch a dialog that allows you to select whether to run the study or launch builder, and if you are running the study it will allow you to choose whether to present stimuli.

This folder will also contain a folder with complete copies of the builder and running scripts, as well as a folder containing all of your stimuli (stimulus files will be copied from their original location, not moved), and a folder where all of your data will be saved. The data folder contains a sub-folder for verbose data files.

Because of this, moving your study between computers is trivial. Depending on the size of your stimuli you could email it, or if not use a flash drive. Best of all, it is trivial to share your experiment on open science platforms like OSF. Simply upload the contents of the folder as-is, or compress it into a zip file, and anyone should be able to download and run it.

There are potential obstacles to this simplicity when it comes to moving between Windows and Mac, or more accurately only when moving from Mac to Windows. For whatever reason, PsychoPy’s movie-playing functionality is much, much better on Mac, and some movie files might not work on Windows machines. Some media files might break, but all the parameters of the study should still be exactly as they were.

An important note about PyHab experiment folders

One of PyHab's most unique features is that when you save a study folder, it doesn't just contain the settings and stimuli for your experiment, it actually contains **all of the code** needed to run it from PsychoPy, **frozen at the moment that the study is saved**.

This means that, if I release updated versions of PyHab, **your existing experiments will be completely unaffected unless you manually replace their code with the updated version**. This has upsides and downsides. One upside is that no matter how much PyHab changes, your experiment will always be exactly as it was when you saved it. That was the goal of making PyHab behave this way. A second upside is that people running your study don't need to download anything other than your experiment folder, PsychoPy, and VLC. They don't need a separate PyHab version of their own.

The downside is that if you want to take advantage of any bug fixes or new features, you have to manually replace the code in your experiment's PyHab folder with the new code. This isn't too difficult. In your experiment folder, there is a folder named "PyHab". This contains a handful of Python files (ending in '.py'). If you want to update them, simply take the files of the same name from the most recent version of PyHab, and replace the ones in your study folder. However, while I have done my absolute best to make sure PyHab is always backwards-compatible, **I cannot guarantee that your experiment will work if you do this**. In some cases, it may be better to simply remake your experiment from scratch, rather than try to upgrade it in this way.

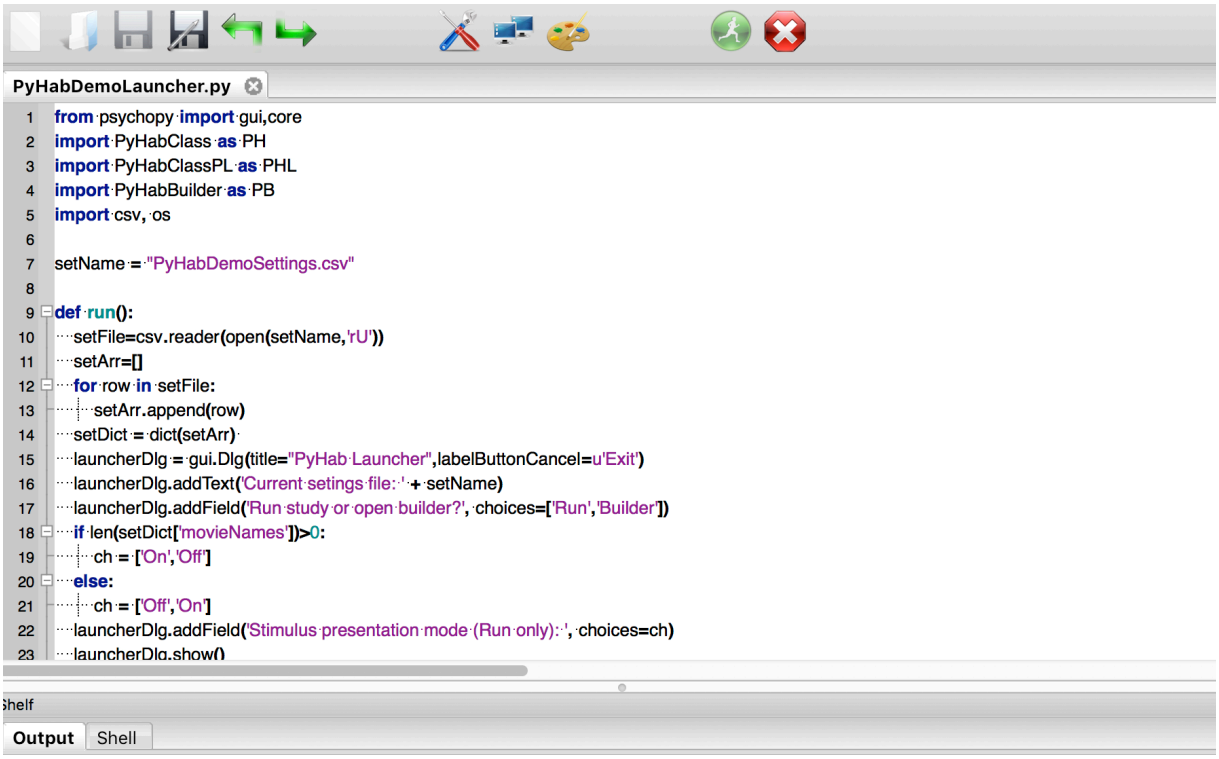
4. Running a study in PyHab

You (or your PI) built a study in PyHab and now you're ready to run some participants! These instructions are meant to be accessible to anyone, whether you're the one who designed the study or an RA helping to run it. Once the study is built, this is everything you need to know.

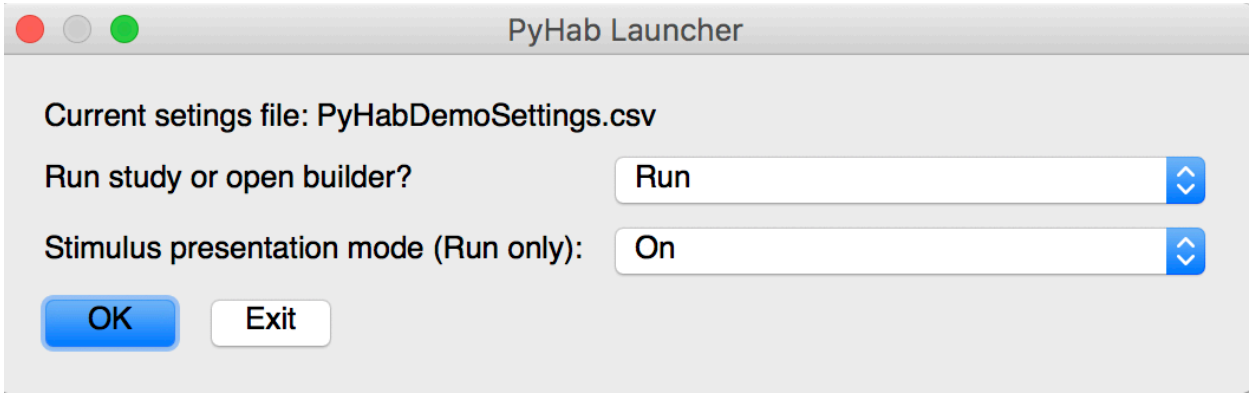
a. Starting a PyHab study

First, open PsychoPy and make sure you are in Coder view. Then, in PsychoPy, open the "[studyname]Launcher.py" script located in the study folder. There will be a bunch of other .py scripts in this folder. You can ignore them.

Once you have opened the script, your window will look something like this.

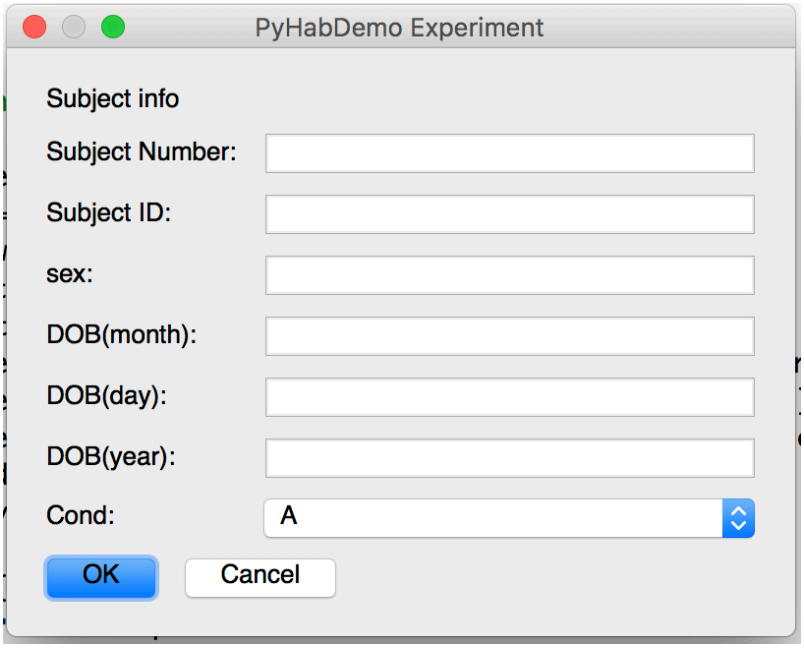


Simply click the green "Run" button in the top right. This will open a dialog that looks like this.



The first line specifies whether you want to run the study or open up the builder view to modify it. Most of the time you will have it on “Run”. The second line is whether or not you want PyHab to present stimuli. This can be set to “off” for offline coding or for studies in which PyHab is not responsible for the stimulus presentation (for example, if your stimuli are a live puppet show). When you are ready, hit “OK”.

When you hit OK, after a few seconds, a dialog box will pop up. There are a few different ways it can look, but the simplest one looks like this:

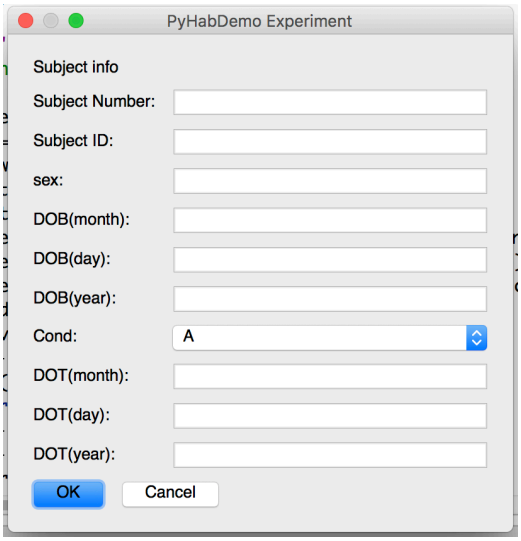


Here’s what each of those fields does:

- The subject number and subject ID go into how the data files are named. Importantly, if you have already run a participant with the same subject number and ID, it will create a duplicate data file with a number after the subject ID. **You must put something in at least one of these fields or the program will crash.**
- Sex is just for the data file. It can be left blank.
- The DOB fields are for the participant’s Date of Birth and are **required**, and PyHab will give you an error message if you leave it blank. DOB is used to compute their age at the time of the study run. You need to input the month, day, and year separately.

Month and day should be in two-digit form, year can be either two-digit[1] or four-digit (and PyHab can tell which is which). So, for a birthday of January 15, 2015, you would put 1 in the DOB(month) field, 15 in the DOB(day) field, and 15 or 2015 in the DOB(year) field.

- Cond is condition the participant is assigned to. It can either be a fill-in-the-blank, as you see in this image, or a drop-down list of different conditions if you set up condition randomization, see condition settings.
- If you are not using PyHab for stimulus presentation, there will be three further fields shown in the screenshot below, that say DOT(month), DOT(day), and DOT(year). These are for recording the Date of Test, i.e. the day on which the study was run. This allows you to re-code data after the fact, and still get an accurate age calculation in the output file. These fields use the same format as the DOB fields. **If any of them are left blank, PyHab will default to today's date.**



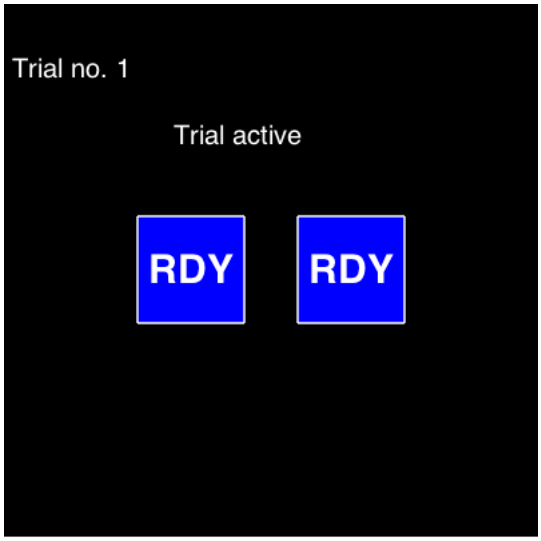
When you've entered all this information, hit OK. This will start the study run.

[1] If you are running this program in the year 2100 or later (over 80 years after its initial creation), the age-at-run computation code will need to be modified for two-digit years. Also holy sh*t.

b. Running a PyHab study

Once you have hit the “OK” button, if PyHab is being used to present stimuli, the stimulus window will appear on the stimulus presentation screen. Whether or not it is being used to present stimuli, a window will pop up on the experimenter screen that looks something like this:

How much information is present depends on the blinding settings, but this is an example of the most informative mode. The trial number is displayed at the top. The trial status is the next line of text, along with the trial type of the next trial. The two boxes are the coding status for two coders (if verbose is set to N, only one box will be displayed). The left box corresponds to the primary coder, the right box is the secondary coder. If you hit the “A” key, it will prepare or begin the first trial, and the display will look like this:



The blue color tells you it is now ready to begin the trial. If the trial does not start automatically (which is an option that can be configured with autoAdvance, see trial settings), the primary coder can start the trial by initiating the first gaze at the screen by pressing the 'B' key.

Depending on the blinding settings, the boxes may show whether each coder is indicating that the participant is looking at the screen or not. When a coder's key is held down, their box will be green and say "ON". When they are not holding down a key, it will be red and say "OFF". When the trial ends, both boxes will go black. With strict experimenter blinding, the boxes will not indicate ON or OFF, but remain blue while a trial is active and turn black when the trial ends.

When the final trial is finished or the "end experiment" key is pressed, the experimenter window will display the text "Saving data…" and then "Experiment finished! Press return to close". When this happens, if there is a stimulus window, it will display the end-of-experiment image you selected or go blank if you have not selected an image. As soon as the experimenter window no longer says "Saving data…", all of the data have been saved. For various reasons having to do with PsychoPy, Python, and certain versions of the major operating systems, it will sometimes get stuck while closing and you will have to hit the red "stop" button to close it down completely. **However, as long as the experimenter window no longer says "Saving data", the data are safe**.

If PyHab ends the study run normally, when you hit return to close the stimulus and experimenter windows, it will go to the launcher window it started with, allowing you to either run it again, go into builder view, or exit.

Key commands

Key	Function
A	Play attention-getter (if PyHab's attention-getter is enabled) and ready trial, or start it if auto-advance is enabled.
B	Primary coder gaze-on (or gaze-on Left for pref. looking). Hold down as long as the infant is looking at the screen, release when they are looking away. If auto-advance is not enabled, the trial will start when this key is pressed for the first time.
L	Single-target only. Secondary coder gaze-on. Just like primary coder, but does not control the flow of the study.
Y	End experiment/quit. For fuss-outs or other situations where you need to end the run early. Immediately saves data and quits the program.
R	Abort/redo trial. If pressed during a trial, ends the trial and marks it as no-good, and then allows the experimenter to restart that trial from the beginning. If pressed between trials, allows the experimenter to redo starting from the **last non-auto-advancing trial**, marking the previously recorded trial(s) as no-good.

S	Skip current trial. Ends the current trial immediately, or skips past it if you have not started it yet. Useful if you want a lot of flexibility as to your trial lengths, or if you want to do something like a calibration where you page through trials quickly.
J	If you are using a habituation design, you can press this between trials during the habituation sequence to immediately jump to the test trials. In other words, it behaves as though the habituation criterion was met even when it wasn't.
I	If your are using a habituation design and have reached the end of the habituation sequence (because the criterion has been met or because of the number of habituation trials presented), this key will insert another habituation trial or block.
P	Print trials so far to the output window in PsychoPy. Only works when not presenting stimuli, and only works in-between trials. Will not work if auto-advancing.
M	Preferential looking version only. Gaze-on Right. Preferential looking only supports one coder, so this key can also mark the beginning of the first gaze-on to start a trial. In every way identical to the B key, except indicating that the participant is looking at the other side of the screen.

c. Data and computing reliability

Data

PyHab saves up to four different .csv files at the conclusion of a study:

- Summary data file: Each line is one trial. Columns are determined by the data settings and can be in whatever order the settings specify. The list of all possible settings is:
    - sNum: Subject number (as entered when you start the study)
    - months: Participant age in months; days, days is in the next column.
    - days: Participant age in days above and beyond the age in months.
    - sex: Participant sex
    - cond: Condition. If a condition file is used, this will correspond to the right column of the condition file for that participant's condition. Otherwise it's whatever is entered in the "cond" field.
    - condLabel: If a condition file is used, this will correspond to the left column, i.e. the condition selected. If a condition file is not used, it will be the same as cond.
    - trial: Trial #
    - GNG: Good/No-Good. 1 if the trial is usable, 0 if the trial was aborted or redone. Usually 1.
    - trialType: The trial type
    - stimName: If stimuli are being presented, this will correspond to the name of the movie file for that particular trial.
    - habCrit: Habituation criterion, if applicable. If the study does not involve a habituation sequence, this will be 0. If it does, it will be 0 until the trial on

which the criterion is calculated, at which point it will be the habituation criterion.

- habTrialNo: If you have a multi-trial habituation block, this will tell you what iteration of that habituation block each trial belongs to.
- sumOnA: Total gaze-on time in the trial, as coded by the primary coder.
- numOnA: Number of gaze onsets in the trial, as coded by the primary coder.
- sumOffA: Total time looking away in the trial, as coded by the primary coder.
- numOffA: Number of gaze offsets in the trial, as coded by the primary coder
- sumOnB, numOnB, sumOffB, numOffB: As above, but for the secondary coder.

- Block and habituation summary files: Identical to the above, except that multiple trials are compressed into a single line. For the habituation summary file, each line from the habituation block compresses a single iteration of the habituation block, but only adds up individual trials that were included in the habituation calculations. For the block summary file, blocks you select will be compressed to a single line (see Block data settings).
- Verbose data file A: The primary coder's "verbose" data file. The verbose data file is very similar to the summary file, except that instead of recording summary statistics for each trial, each row is a single gaze-on or gaze-away. Some of the columns are identical to the ones in the summary data file, but there are four new ones:
  - GazeOnOff: Indicates whether the row is showing a time when the gaze-on button was held down, or a time when the participant was looking away. 1 if gaze-on, 0 if gaze-off.
  - StartTime: The time at which the gaze-on or gaze-off began, relative to the start of the trial.
  - EndTime: The time at which the gaze-on or gaze-off ended, relative to the start of the trial.
  - Duration: The difference between EndTime and StartTime.

- Verbose data file B: If there are two coders, and the secondary coder inputs at least one gaze, a second verbose file will be created for the secondary coder. It is identical to the verbose data file for the primary coder.
- Reliability: If verbose data file B is created, then PyHab will also attempt to compute reliability statistics comparing the two coders, which will be output to this file. There are four reliability statistics:
    - Weighted percentage agreement: Percentage of frames on which the two coders agreed, weighted by the length of the trial.
    - Cohen's Kappa: A reliability statistic between 0 and 1, preferred by some journals and researchers.
    - Average Observer Agreement: The raw percentage of frames on which the two coders agreed.
    - Pearsons R: Standard correlation coefficient.

Standalone reliability

There is also a stand-alone reliability script which allows you to take any two verbose data files and get the afore-mentioned reliability statistics. When run, the script asks for information about the subject, and when you hit OK, it will open two file-open dialogs, one after the other. Select the two verbose data files you want to compute reliability for, and it should save a reliability .csv in whatever folder the script is found in. It will also print the results to the PsychoPy output window.

d. Preferential looking studies

PyHab has a variant for preferential looking designs, in which infants have two possible targets they can look at and you want to code which one they are looking at, not just whether they are looking at the screen. Running a preferential looking study is *almost* identical to running any other study in PyHab, with a few important differences:

- The preferential looking version only supports one coder. You cannot have two people simultaneously coding a preferential looking design (just not enough keys on the keyboard). However, you can still have one person do live coding and another person do off-line coding and then get reliability using the standalone reliability script.
- The 'B' key is now "gaze-on Left". The 'M' key is "gaze-on Right". They work completely identically except that one indicates a gaze to the left and the other indicates a gaze to the right. Either can be used to start a trial, time spent looking away (for determining when to end a trial) is only computed when neither button is being held down.
    - If your study has some single-target trials and some preferential looking trials, just use one of the keys for the single target trial(s).
- In the data, the looking-time columns will now be "sumOnL", "numOnL", "sumOnR", "numOnR", "sumOff", and "numOff". No more A and B (because only one coder). As you might expect, if it says L it refers to gazes to the left, if it says R it refers to gazes to the right.
- The verbose data file now has three codes in the gazeOnOff column: 0 = "off", 1 = "gaze-on Left", 2 = "gaze-on Right".

Everything else is the same between the two versions.

5. Troubleshooting

PyHab is far from perfect. Development will continue for some time. Let me start by pointing out a few things you don't need to worry about: Every time you run PyHab, there will be some number of messages in PsychoPy's "Output" area at the bottom of the coder interface. If PyHab crashes unexpectedly, there will probably be some blue text here telling you the nature of the crash. Even when it runs successfully, there will be some green text that says WARNING or brown text that says ERROR. Don't worry about those. They don't affect the functionality of PyHab, they're just quirks of PsychoPy and some of the libraries PsychoPy uses.

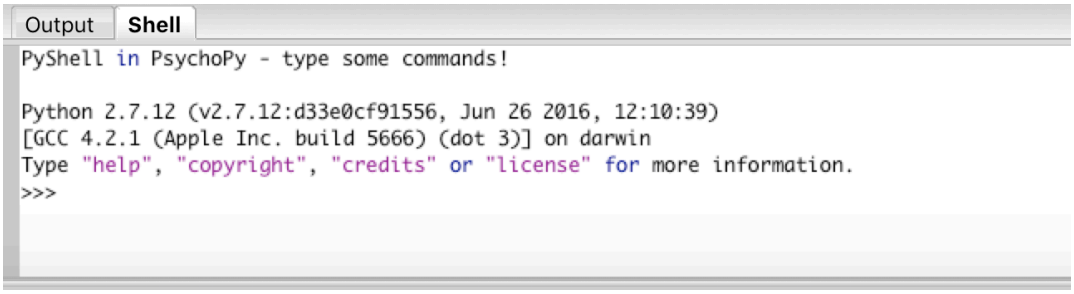
As for everything else, here are some known issues you might encounter:

- **Experimenter and stimulus windows appear and everything seems to load OK, but keys don't work**: This is most likely because the "focus" is not on the experimenter window. Just click on the experimenter window (anywhere in it) and it should start responding to the keys.
- **Program crashes right after pressing the green button, says something involving "segmentation fault"**: This is an issue that occurs mostly on Macs, and it's not a PyHab problem. It's just a known issue with PsychoPy2 and MacOS, and it's relatively harmless. If you are using PsychoPy3, you shouldn't encounter this at all. Just hit "ignore" on the the error dialog that pops up and try running it again. It sometimes takes two or three tries but it does eventually work.
- **The stimulus presentation window opens halfway between the two screens:** This is a problem that seems to mostly occur on Macs with Retina displays. It breaks differently in later version of PsychoPy, but the bottom line for the time being is that using a retina display with an external monitor won't work reliably. I am actively trying to figure out how to fix this in PsychoPy itself (the nice thing about open-source software is that anyone can contribute to it), and I've recently been able to isolate the cause, and I'm working with the PsychoPy devs to try to figure out a solution. In the most recent versions of PsychoPy, some people have had success making the external monitor the

primary display (under system settings -> displays, go to “arrange” and drag the white bar over to the external monitor), but it’s not a consistent fix unfortunately.

- **When running PyHab, the program freezes on “Loading Movies” and crashes**: This is most likely to occur on Windows, and it basically means your movie files are too big or have too high a bitrate for PsychoPy’s movie system to handle. The only fix for this is to modify the movie files themselves. Since you need VLC anyways, you can open the movies in VLC and use it’s “save/convert” function. Try converting your movies to MP4 and, if possible, reducing the resolution or framerate slightly. Hopefully future versions of PsychoPy or the underlying moviepy code will make this less of a problem.
- **At any point, it crashes and displays a “permission error”**: This is usually a Windows problem. Make sure you run PsychoPy as an administrator (right click on the icon and select “run as administrator” or right click and go to properties and change the settings to always run as administrator). PsychoPy needs to control a lot of very low-level features of your computer, and your operating system won’t allow that unless you tell it to.
    - If something like this happens on Mac, go to your system security settings, “Accessibility”, and add PsychoPy to the list of programs that are allowed to control your computer.
- **When running PyHab, the program crashes immediately on “Loading Movies”**: If it does this once, try running it again. If it does this every time, check the “Output” area at the bottom of the PsychoPy coder window. If it says something about FFMPEG, it means you’re missing a codec. **On Mac**, just install VLC media player onto your computer. **On Windows**, there can be a specific codec that you need to install in PsychoPy itself, but this is not too difficult and you only need to do it once:
    - Run PsychoPy as an administrator (Right click the icon when you start the program)

- In the lower half of the coder view, there are two tabs, “Output” and “Shell”. Click “Shell” and you will see something like this:



- At the >>> prompt, type (without quotes) “import imageio” and press enter
- When the prompt re-appears, type (without quotes) “imageio.plugins.ffmpeg.download()” This will cause a bunch of text to appear, let it run until the prompt re-appears again.
- Click on “Output” again and you should now be all set to run!

- **PyHab runs the experiment successfully, but hangs at the very end and does not show the launcher menu, but the red stop button remains highlighted and the green button is grayed out**: This is another PsychoPy2 issue that should be much less common in PsychoPy3. Again, more common on MacOS than Windows. In either case, just hit the red stop button. In either case, as long as the experimenter window no longer says “Saving data”, your data have been saved, so there’s no risk in doing this.
  - Occasionally, under conditions I haven’t been able to figure out, this will happen while saving the reliability file. If it hangs on “Saving data” for a long time (like several minutes) after a single-target run with two coders. After a few minutes, I recommend stopping it manually, and it should save the essential data files.
- **The program crashes on load or on a specific trial type while using condition randomization, or the builder crashes when loading the condition settings screen**: Basically if you change something about the way stimuli are assigned to trial types *after* creating your condition file, and don’t go back and change your condition file as well, PyHab can get confused. If going into the builder and changing the conditions there doesn’t work, close PyHab and do the following:
  - Delete the “conditions.csv” file in your study folder.

  - Open “[studyname]settings.csv” in something OTHER THAN excel, like Notepad or BBEdit (Excel will reformat some things in a way that breaks PyHab).
  - In the settings file, look for three lines: “condList”, “condPath”, and “condFile”. Erase anything after the comma. For “condList", put “” after the comma.
  - Save the settings file and re-open PyHab in the builder to make a new set of conditions.
- **After you modify an existing experiment by adding a new trial type, it crashes on load every run.** If you add a new trial type to an experiment with an existing condition file or existing condition settings, the new trial type won’t be automatically added to the new condition file. You need to go in manually and modify your existing conditions, or create a new condition file from scratch, that has the stimuli you want to use for that trial type. PyHab should warn you to do this when you create the new trial type if you already have an existing condition file.
- **Any other crash**: If it shows any blue text in the “output” pane of PsychoPy, copy as much of it as you can and send it to me at jonathan.f.kominsky@gmail.com with a description of what you were doing when it happened, and I’ll try to figure it out. You can also create an “Issue” in the GitHub page at https://github.com/jfkominsky/PyHab/issues