Pyhab User Manual

v0.10.4

Fall 2022

Dr. Jonathan F. Kominsky

1. What is PyHab? 3
2. Getting PyHab 4
What do you need? 4
Installation instructions 6
3. How to build your study in PyHab 8
a. Relevant concepts 8
b. Creating your first experiment 10
Trial types and blocks 13
Making your study flow 17
Adding stimuli to the experiment library 19
Assigning stimuli to trial types 22
Advanced trial settings 26
Blocks 29
Habituation 33
Deleting trial and block types 36
Settings windows 37
Universal settings 37
Stimuli settings 39
Habituation settings 40
Attention-getters 42
Data settings 44
Condition settings 46
Head-turn preference procedures 58
c. Saving and sharing your experiment 64
An important note about PyHab experiment folders 64
4. Running a study in PyHab 67
Introduction 67
a. Starting a PyHab study 68
b. Running a PyHab study 72
Key commands 75

c. Data and computing reliability 78
Data 78
Standalone reliability 81
d. Preferential looking studies 82
e. Head-turn preference procedure 83
f. Using PyHab with a Tobii eye-tracker 84
5. Online studies 87
a. Basics and warnings 87
b. Extra installation steps 90
c. Building an online experiment 92
d. Running an online experiment 95
6. Troubleshooting 97
Movie stimuli freezing or playback issues 100

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. During the COVID-19 pandemic, I added a way to run studies online without relying on screen-sharing as well.

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. For most cases, you want to use the most recent version of PsychoPy, which you can download from here: https://www.psychopy.org/download.html
- **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. Lately, mp4 has been the most reliable. 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. Alternately, there is a setting that makes it so you should never have to rewind a movie file at all, it just loads each movie file separately for each trial, but this eats a lot more RAM.

- 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 may 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).

- **Citing PyHab:** If you use PyHab in your experiment, **please cite both it and PsychoPy**. The relevant citations are:

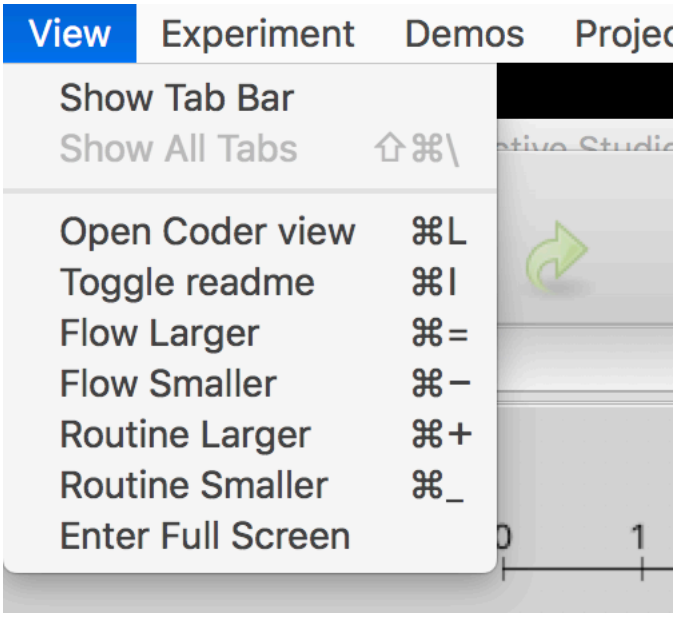
Kominsky, J. F. (2019) PyHab: Open-Source Real Time Infant Gaze Coding and Stimulus Presentation Software. *Infant Behavior & Development, 54*, 114-119. doi:10.1016/j.infbeh.2018.11.006

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

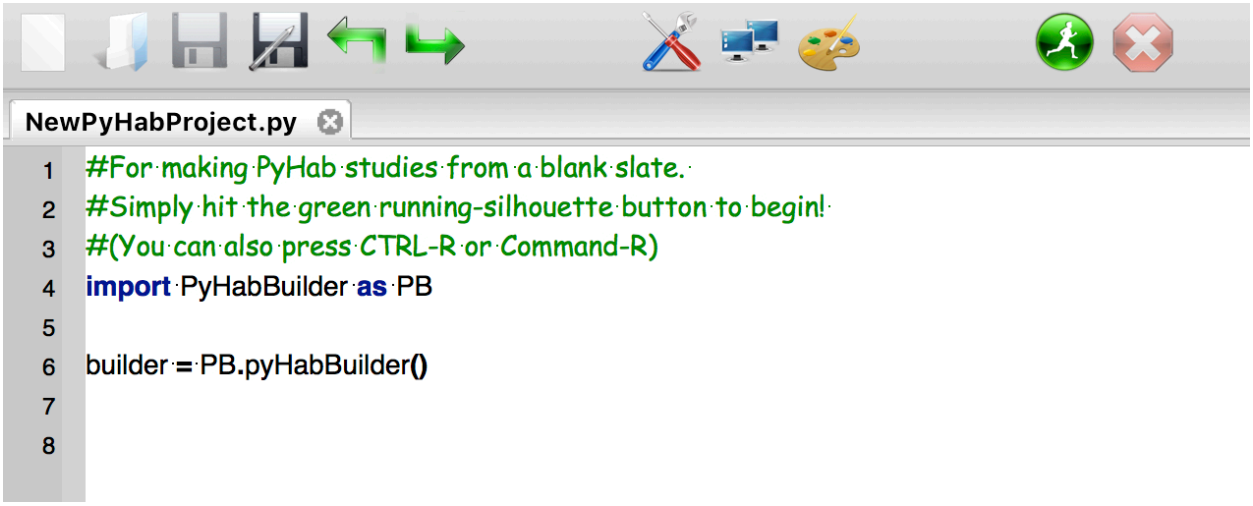
Installation instructions

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

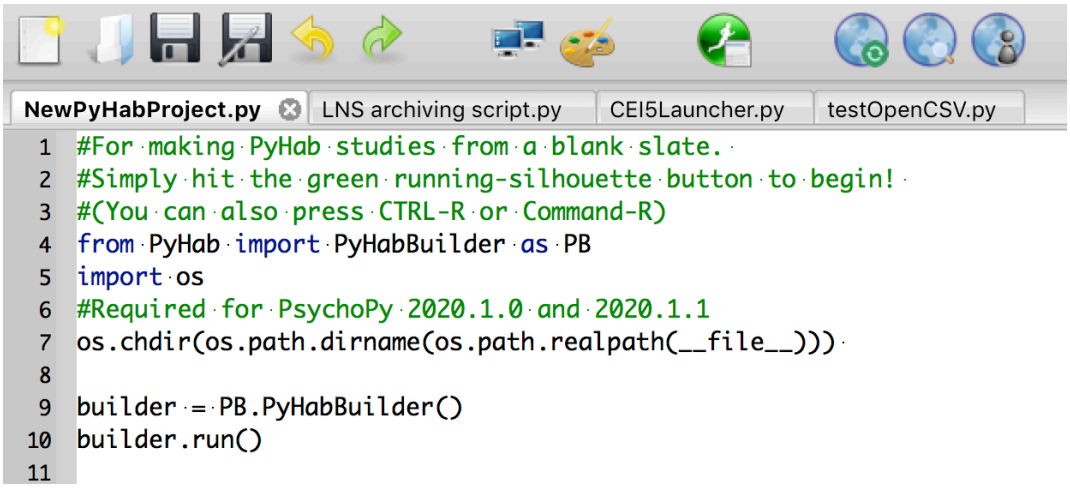
Let's define a few important terms:

- **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 any of these.
- **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.
    - Habituation can only be done using blocks. A block can consist of only one trial, but the habituation setting are controlled at the level of the block. For more information see Habituation.

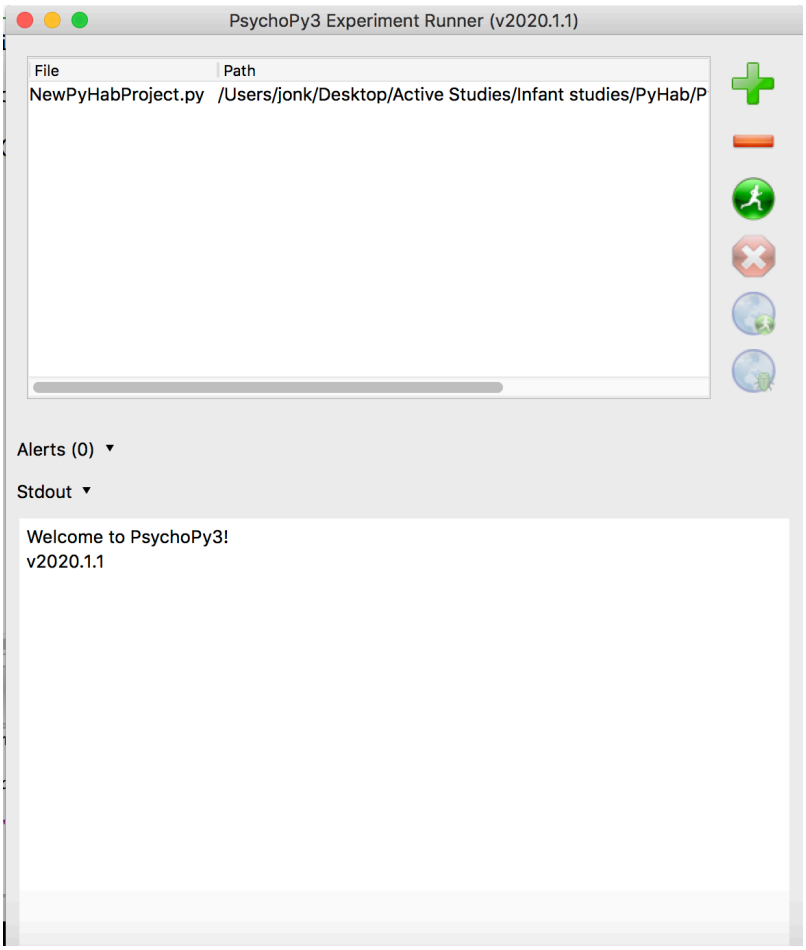
- **Stimulus file/stimuli**: PyHab expects all stimuli to be in the form of movie, audio, or image files, which you import into your PyHab experiment from the builder. You can also run PyHab just as a looking-time coding software with no stimulus presentation, in which case no stimulus files are needed.
- **Stimulus library**: When you add media files to PyHab, you create a "library" of files which can then be associated with different trial types.
- **Conditions**: A predefined presentation order of stimuli for a given experiment, that can be selected for each individual participant. Can also be used to control order of trials within a block!
- **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).
- **Head-turn preference procedure or HPP**: A vastly more complicated type of infant looking-time study involving up to three different screens. If this is relevant to you, you know what it is. PyHab works for HPP studies, but there are a couple of extra steps.

b. Creating your first experiment

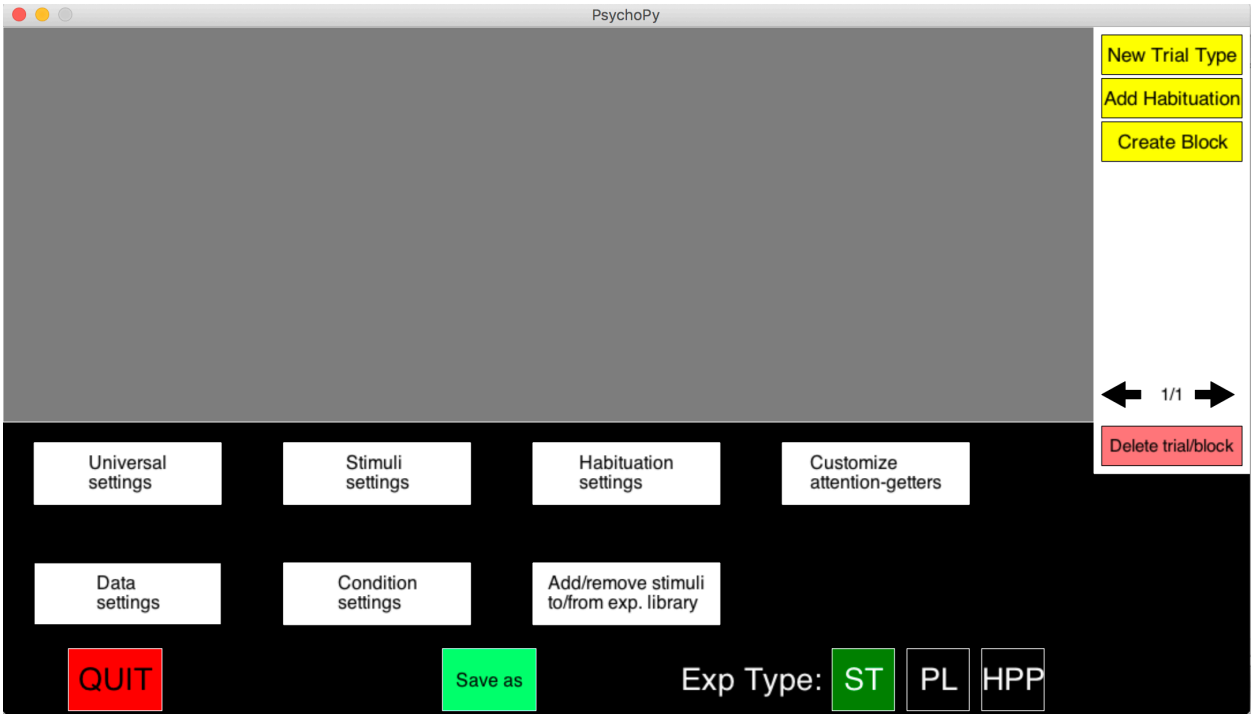
Open the “NewPyHabProject.py” script in the PsychoPy coder window (see Chapter 2). You will see something like this. Hit the green running man icon on the right to launch the builder window.



In PsychoPy 2020.1.0 and later, a new window will open that looks like this. Simply hit the green running man button again.



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



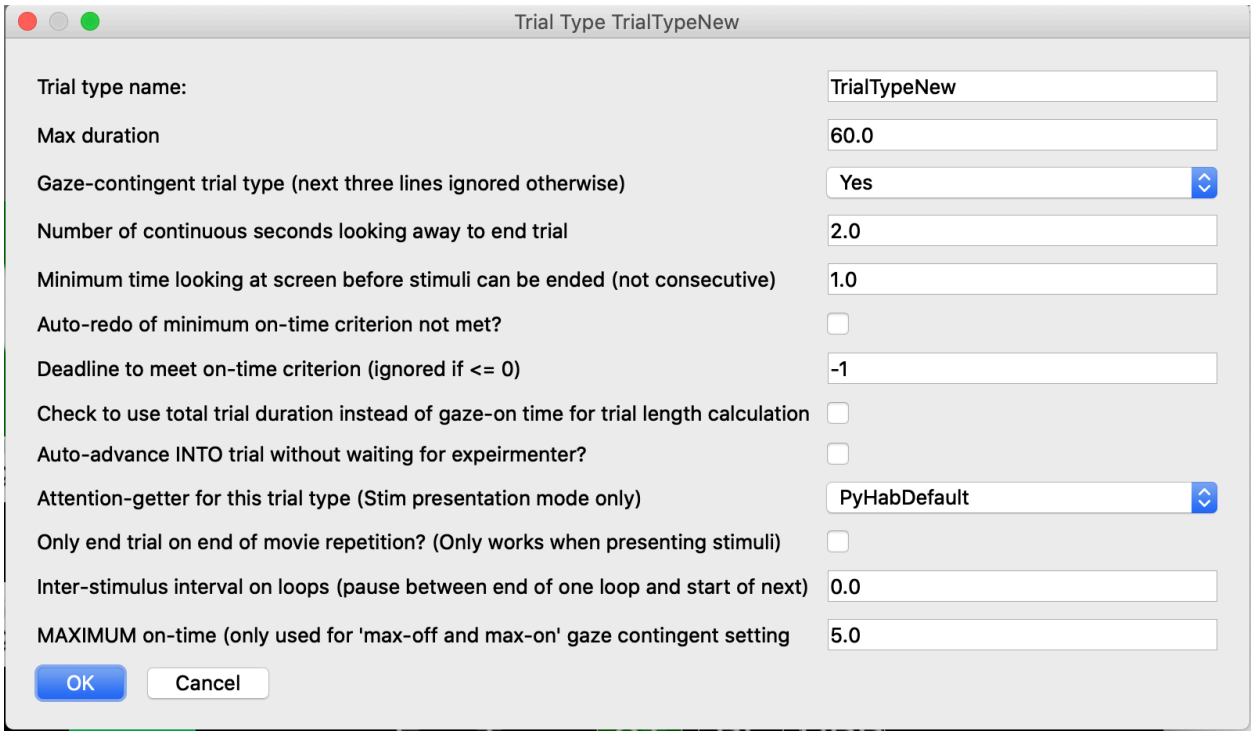
This is your main builder view, and what you will be dealing with most of the time. The top area with the grey background is the study flow, which will ultimately show you the order in which your trials will be presented. The area on the right is the trial types palette, where you can define new trial types for your study, edit existing trial types, or delete trial types (and also do all of this for blocks made up of multiple trial types). All of the other settings can be accessed through the six buttons in the lower half of the screen.

Most 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 criteria are met. If it is set to “Either/or”, it will end when either the minimum-on OR look-away criterion is met, whichever comes first. If it is set to “MaxOff/MaxOn”, it will function like the normal gaze-contingent trial (ending when the minimum on and look-away criteria are met), but the trial will also end if the infant has looked for the maximum on-time (the last line in the window).

The minimum-on and maximum-off criteria for the trial type are set in the next two lines (the maximum-on criterion is set in the final line). 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.

Next is a check-box to allow the trial to automatically redo itself if the on-time criterion is not met by the time the trial ends. If this is checked, PyHab checks at the end of the trial whether the total on-time is less than the minimum on-time, and if it is, it restarts the trial. Following that is an additional auto-redo feature, which is the “on-time deadline”. If this is set to a value greater than 0, then PyHab checks whether the minimum on-time has been met *by that time in the trial*. So, for example, if you put “5” in this box, PyHab would check at the 5-second mark whether the infant has looked for the minimum on-time, and if not, immediately end the trial and restart it. This overrides any other end-trial criteria, including “end on movie end” (see below).

The next check box allows you to use trial duration (including on-time and off-time) rather than gaze-on time to determine whether a minimum on-time has been met. However, this *does not work* with the auto-redo functionality, this is only for determining whether the on-time has been met for ending the trial normally. Auto-redo *always* uses actual gaze-on time, no matter what.

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.

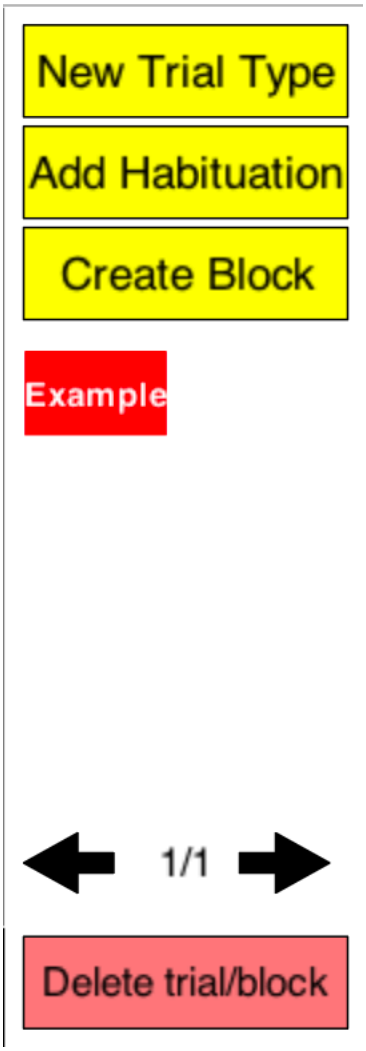
> **Tip:** If you want a non-gaze-contingent trial to play a movie exactly once and then end, check this box and set the "max duration" value to less than the length of the movie. This checkbox takes precedence: It will play to exactly the end of the movie and then end the trial.

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).

Finally, the maximum on-time is only used if the trial type uses the "MaxOff/MaxOn" gaze criteria and is ignored otherwise. If you are using this gaze criterion, this is how long the infant has to look to end the trial (if it hasn't ended for any other reason first).

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).

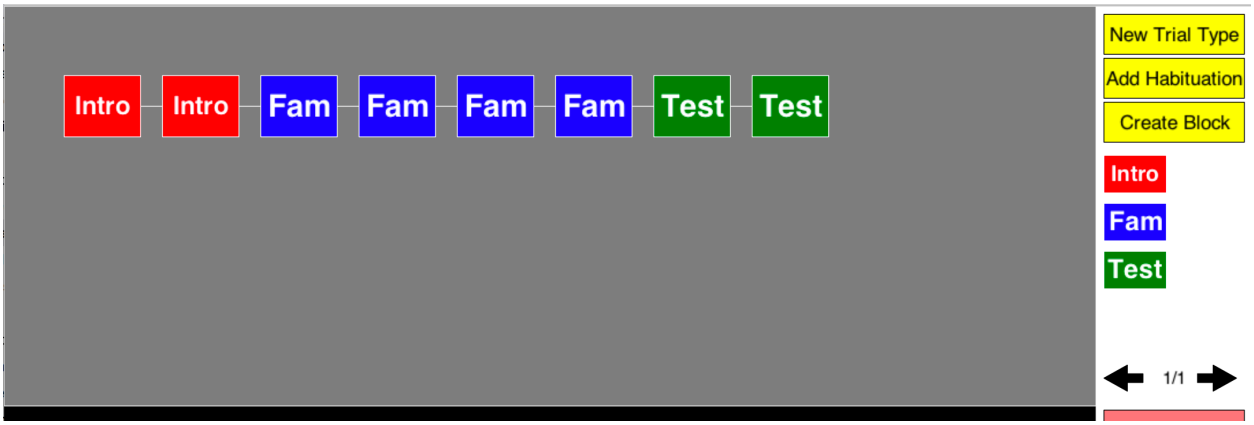
We'll go through each of these in turn.

Making your study flow

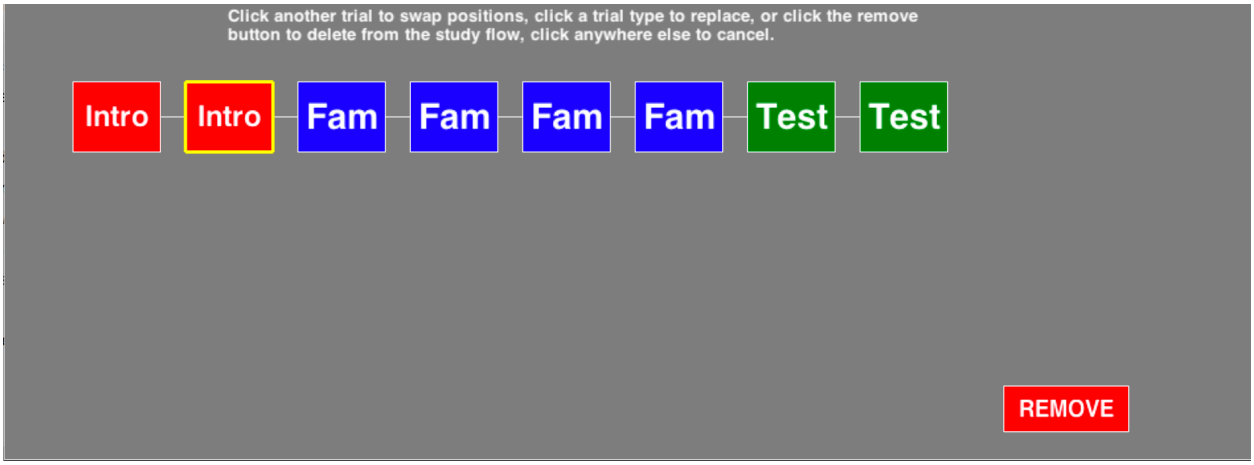
The "study flow" is the order in which different trial types will be presented. Say you have three trial types, intro, familiarization, and test. Your palette might look something like this.

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

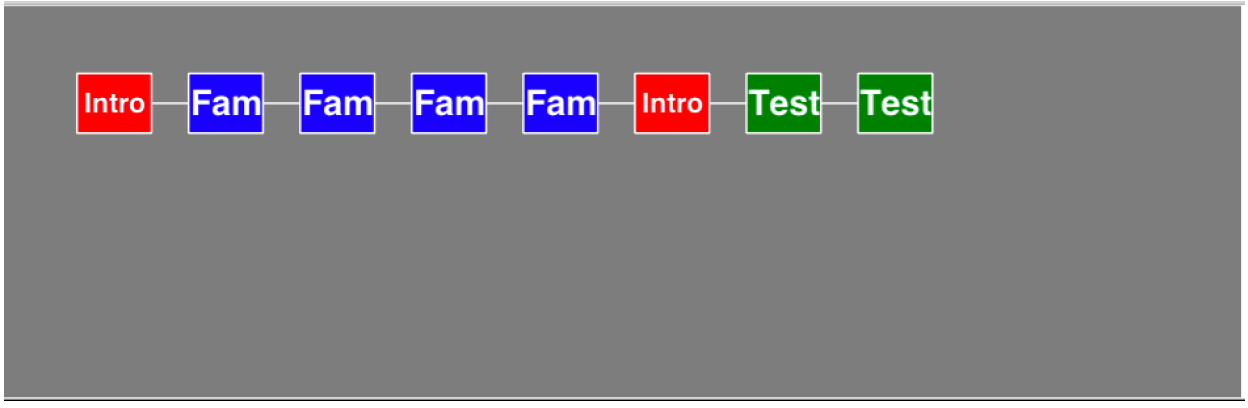
“test” button twice. This would leave you with a study flow that looked like this.



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.



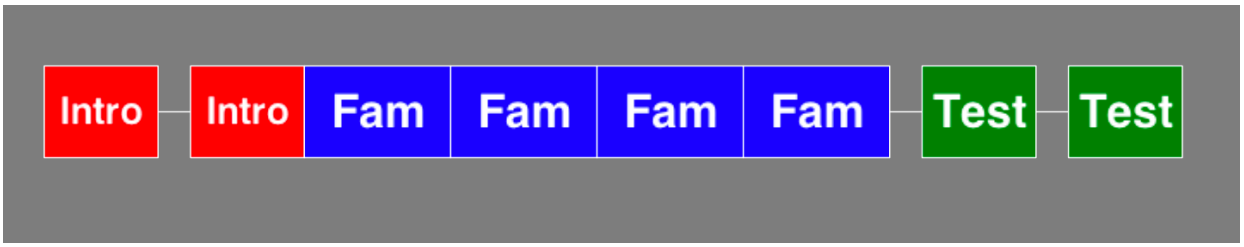
The instructions at the bottom are pretty self-explanatory. To move the intro trial to after the last Fam trial, simply click on the last Fam trial and they will switch places.



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).

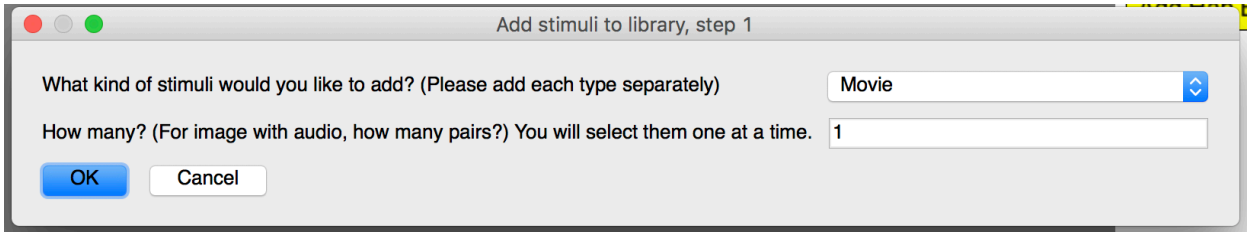


Note that this is ALL you can do with the study flow directly. If you want to change the order of stimuli within a trial type, or change the settings of a trial type, all of that has to be done in their respective menus.

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:

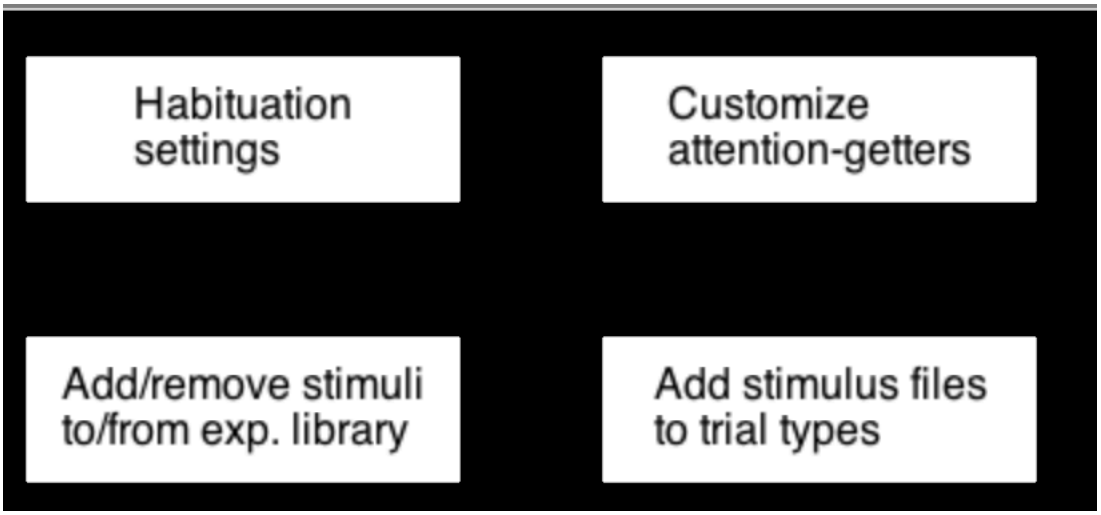


The first line lets you select what kind of stimuli you want to add. The second line specifies how many stimuli of that type you want to add. You can only add one type of stimuli at a time, but you can add as many of that type as you want. There are four types of stimuli you can use in PyHab:

- **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.

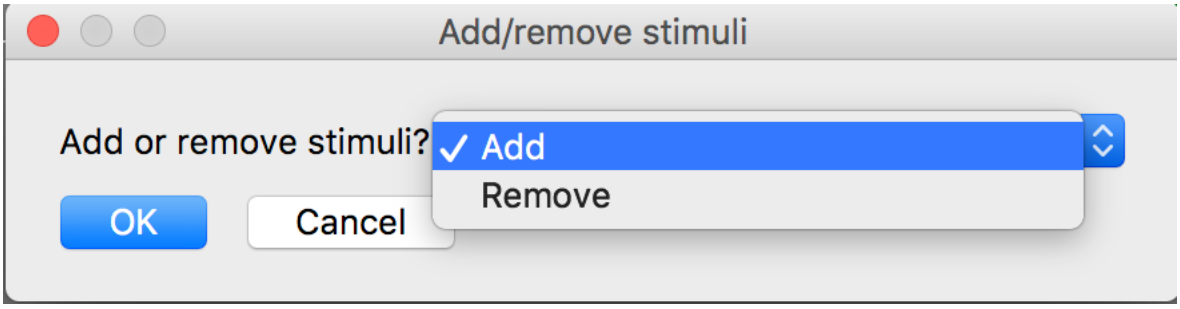
When you hit “OK”, you will then select each file, **one at a time**. For everything but “images with audio”, you will just see the normal “file open” dialog as many times as you specified for how many you want to add. For “images with audio”, you will get extra pop-ups telling you when you are selecting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

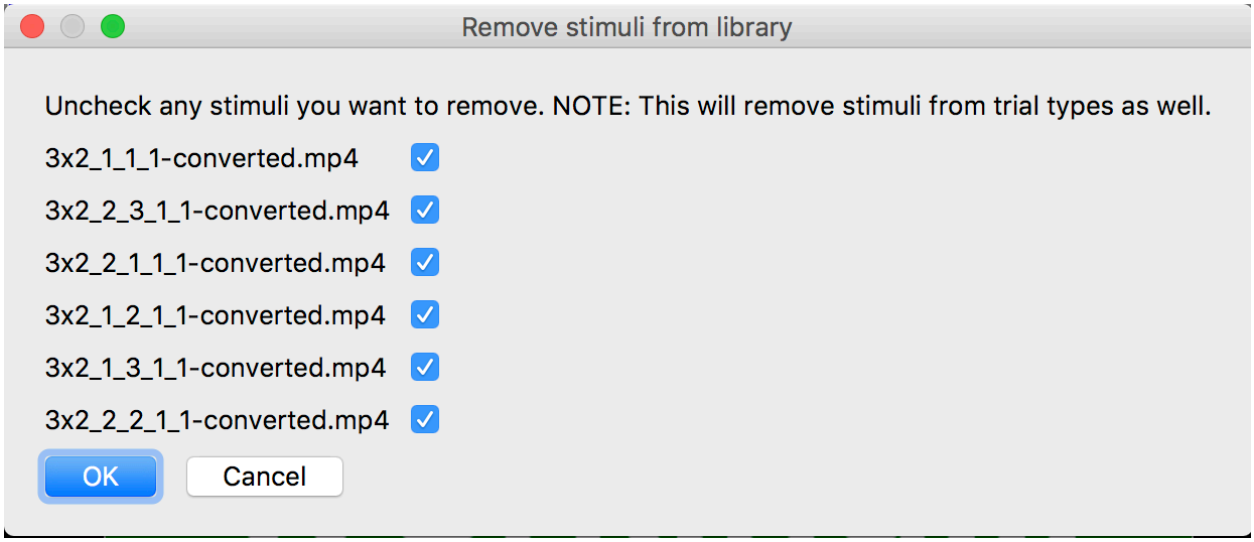


This button allows you to assign stimuli from your library to different trial types, and also add images that will display at the start and end of the experiment.

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.



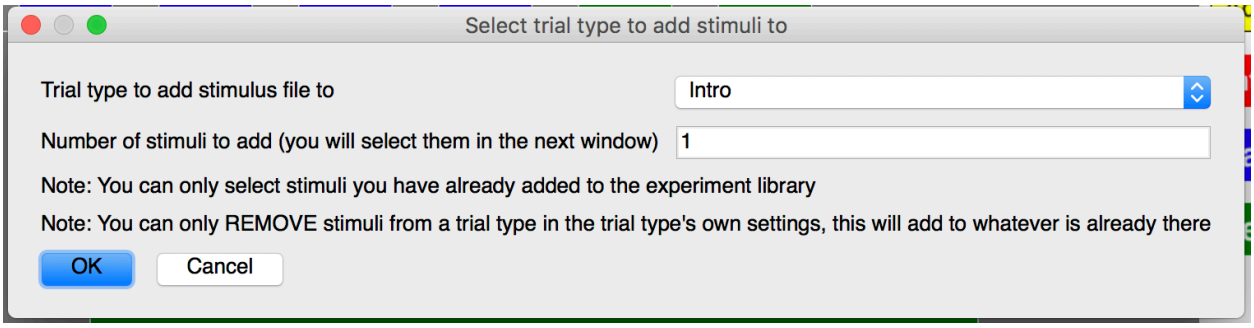
If you select ‘remove’, you will see a dialog like the one below, with a list of all of the different stimuli you have added to your experiment.



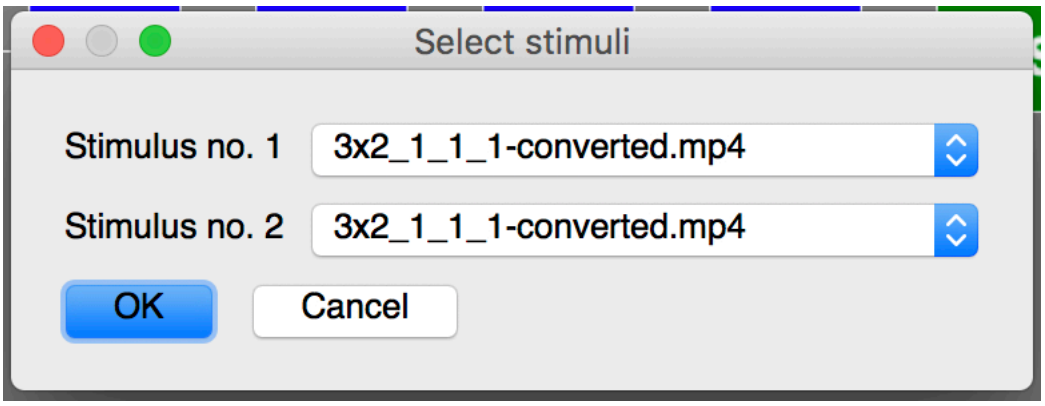
Uncheck any stimuli you want to remove and hit ‘OK’. This will remove the stimuli from the experiment library and from any trial types those stimuli have been associated with. Next time you save the experiment, it will also delete the corresponding stimuli files from the experiment folder (ONLY those 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:



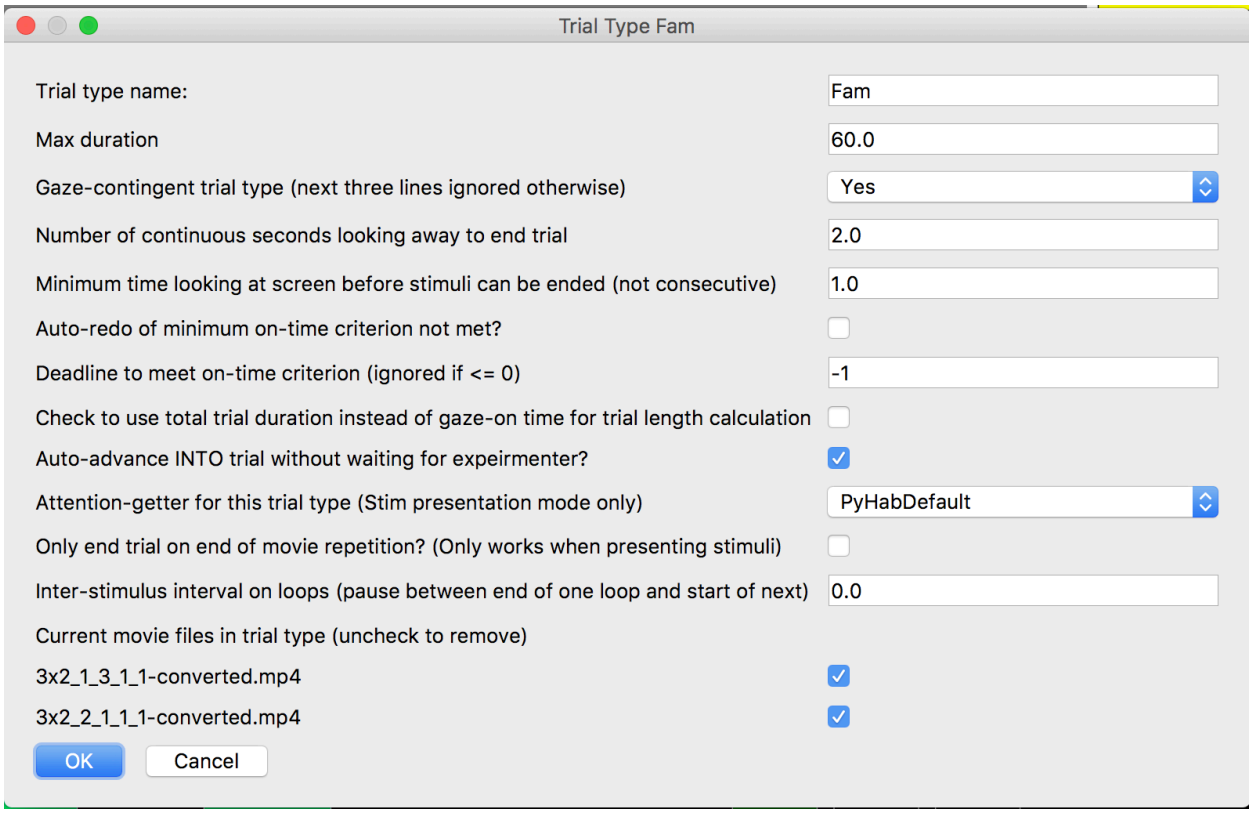
The first line lets you select which trial type you want to assign stimuli to (or if you want to add images to the start and end of the experiment, see below). The second line is how many stimuli you want to add to that trial type. Let's say we want to add two movies to the "Intro" trial type. So, we 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 at the bottom. 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

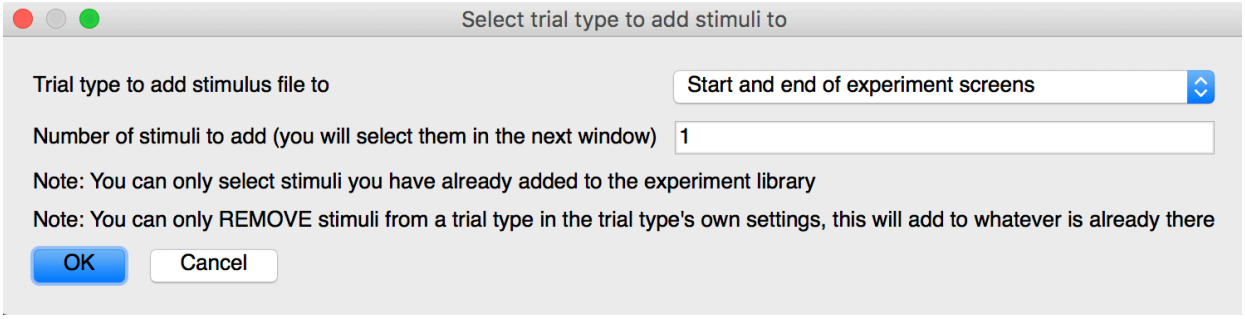
The other thing you can do in this window is add images to the start and end of the experiment. These are completely optional, if you don't add them then the stimulus screen will just be blank (and whatever background color you set in the stimuli settings).

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

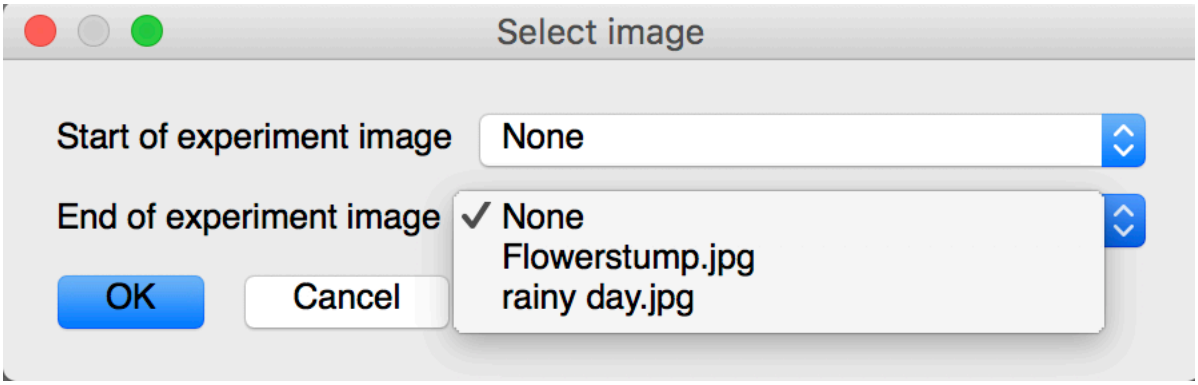
is handy if you want to put something on the screen just to get the infant to look at the screen before you start the experiment.

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.

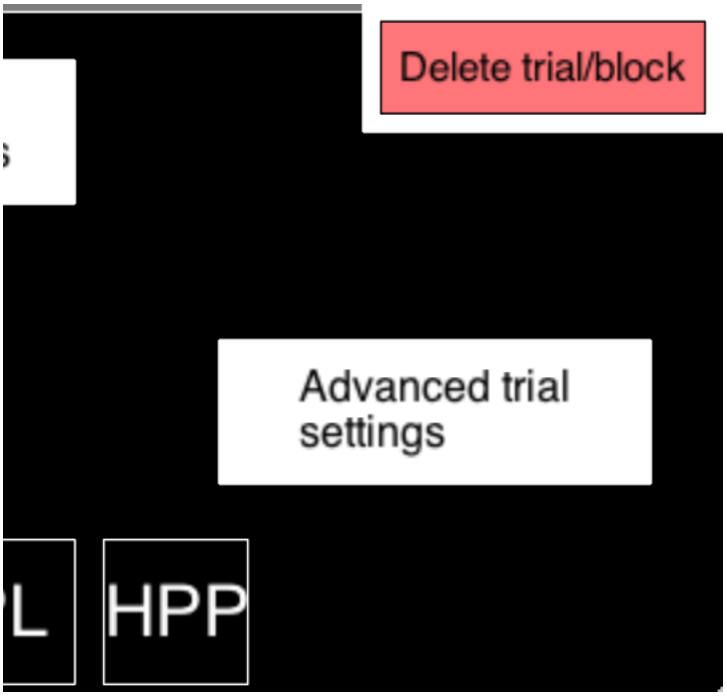


Select the image(s) you want to use and hit "OK".

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.

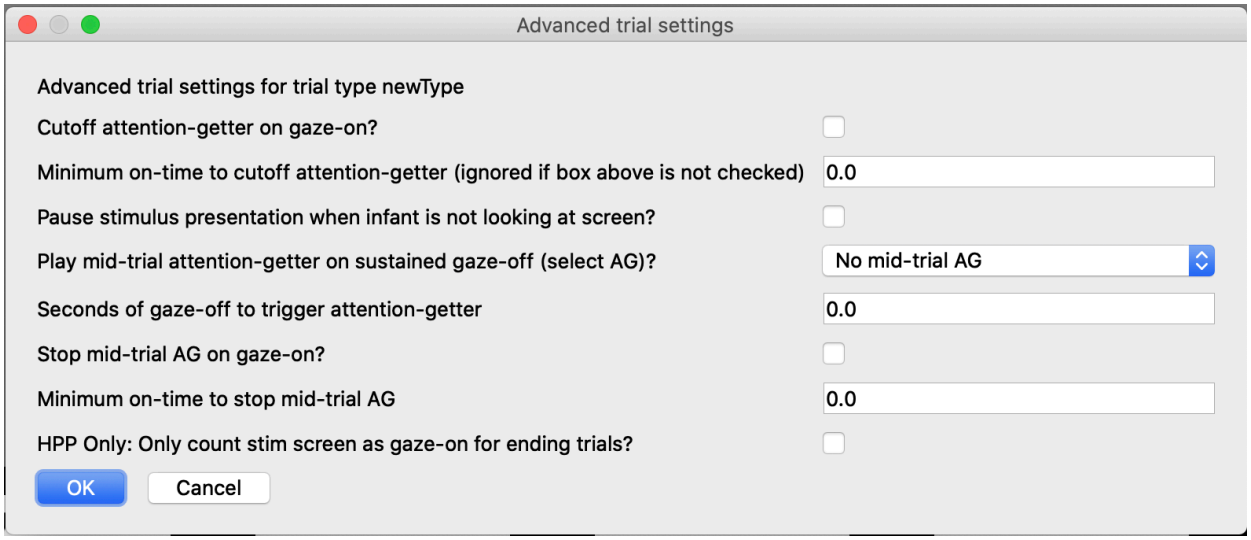
Advanced trial settings

When you create a trial type, even before you add stimuli, a new button will appear in the lower half of the interface:



This button lets you make use of some of PyHab’s more advanced capabilities during stimulus presentation. Click on it, and a new overlay will appear in the builder asking you to choose which trial type you want to access the advanced settings for (note it will only show you trial types, not blocks):

Click on the trial type you want to access advanced settings for and the following dialog will appear:



The first check-box changes the behavior of the start-of-trial attention-getter, if there is one (if there isn't, this and the next line do nothing). By default, attention-getters play all the way to the end, even if the experimenter indicates that the infant starts looking at the screen before the attention-getter is finished. Checking this box means that the attention-getter will end and the trial will start *immediately* as soon as the gaze-on button is pressed. The next line modifies this check-box by adding a minimum time before the attention-getter ends. This is useful if you want to make sure infants see, for example, at least five seconds of your attention-getter before starting the trial, but want to let the attention-getter run longer if they haven't looked at the screen by then. Put in a value in seconds and the attention-getter will always play for at least that long, even if the gaze-on key is pressed.

The next check-box only affects movie or audio stimuli, and makes it so that the stimuli pause if the gaze-on key is not held down. So, if you want to guarantee that infants see all of your stimuli, this will automatically pause the stimulus display until the gaze-on key is pressed again. Note that the trial duration will not be extended! All of the timing settings are independent of this except for ending the trial at the end of a movie. You may need to make the maximum duration of your trial longer, or make it end only on the end of the movie, in the regular trial settings window if you want to ensure the infant sees everything you want them to see. Data will be recorded during this time (it will be recorded as a gaze-off)

The next four settings all have to do with the same thing: Whether an attention-getter plays if the infant has looked away for a certain amount of time during a trial. This goes great with pausing the stimulus display: Say an infant looks away during a critical familiarization, you can pause the movie until they look back, and if they haven't looked back in a certain number of seconds, an attention-getter plays automatically to draw their attention back to the screen. To enable this, select the attention-getter that you want to play from the drop-down menu. In the next line, you can set how many seconds (consecutive) the infant has to look away in order to trip the attention-getter (the default of 0 means that the attention-getter will play the instant the infant looks away). The last two settings are exactly like the first two settings in the dialog, but for this mid-trial attention-getter instead of the start-of-trial attention-getter: They control whether the attention-getter ends as soon as the infant looks at the screen or plays to completion, and whether there is a minimum time that the attention-getter should always play even if they do look back at the screen.

Two important notes: 1. PyHab currently will extend the trial duration by the length of the mid-trial attention-getter. 2. The entire time the attention-getter is playing will be recorded as a gaze-off in the data (even if the gaze-on key is pressed), because the data are designed only to record gaze-on to the *stimuli*. I am trying to figure out a way to record the number of times the attention-getter plays in the data as well without adding a bunch of extra columns to the output that will only be relevant to a handful of people, but let me know if this is a capability you need.

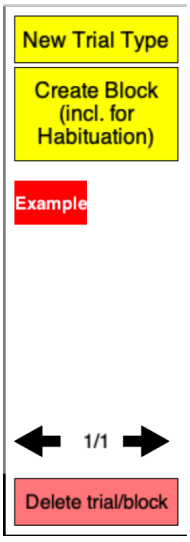
HPP-only setting: "Only count stim screen as gaze-on for ending trials?": The final setting is always visible but only affects Head-turn Preference Procedure studies. You can read more about it in the Head-turn Preference Procedure section of the manual.

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.

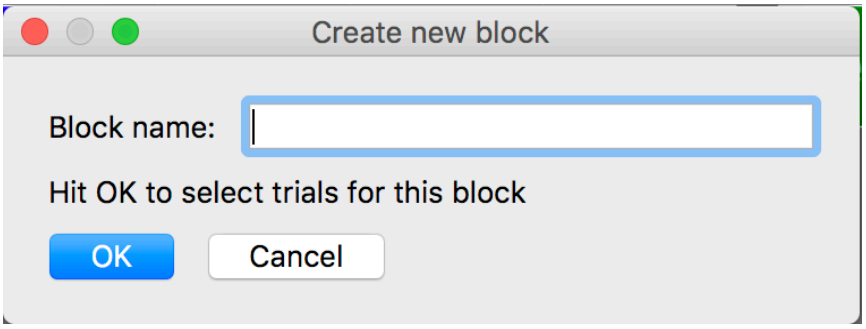
If you don't want to manually add both trial types every time, you can create a **block**. A block is a collection of multiple trials in one compact package. It adds some powerful capabilities to your study design, but let's start with the basics.

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.

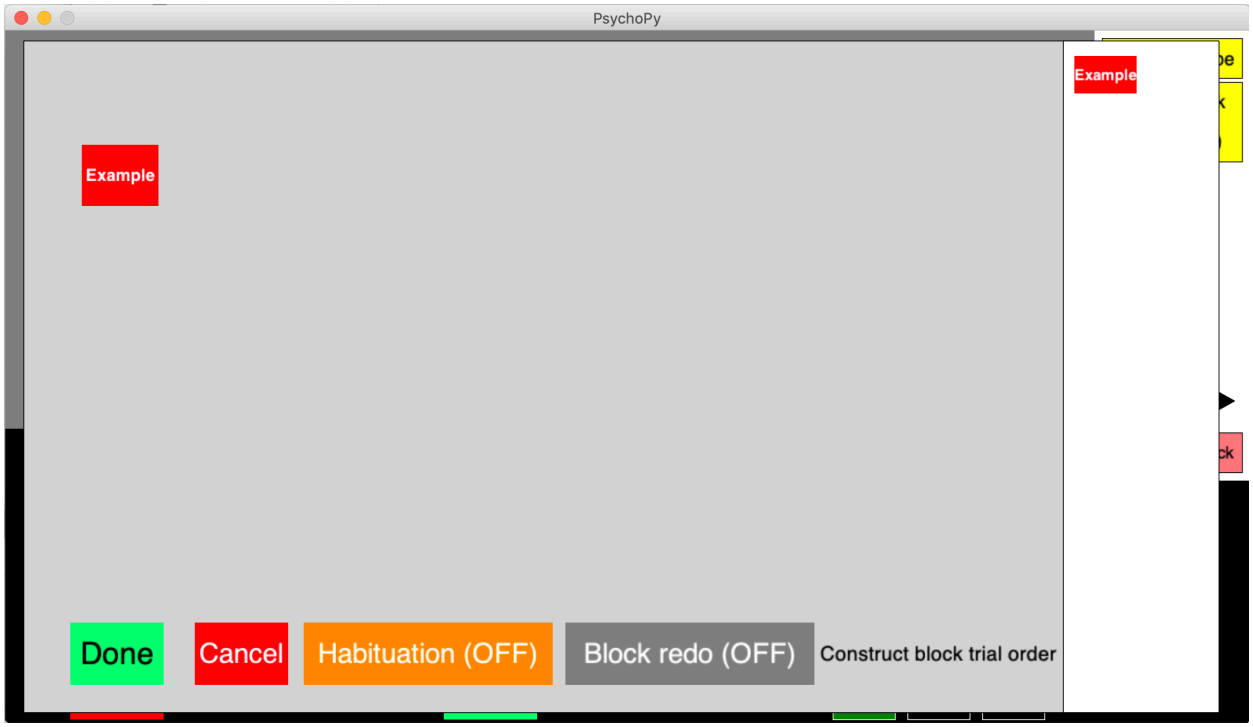


This will open up a dialog that asks you to name the block. This is just like naming a trial type: It's what will show up in the study flow, and in the data files (with a twist - see below). The same restrictions that apply to trial type names also apply to block names: They must be unique,

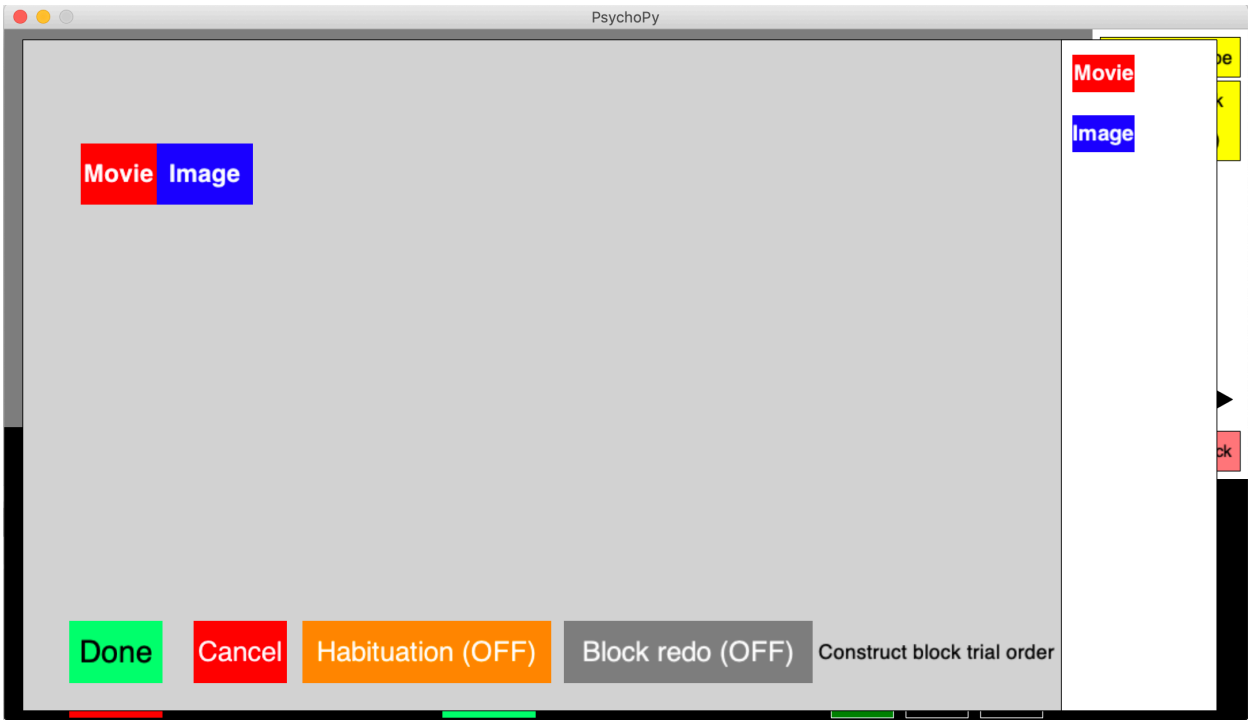
they cannot be ‘Hab’, and certain characters are excluded. You will get a warning dialog if the block name you provide is invalid.



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

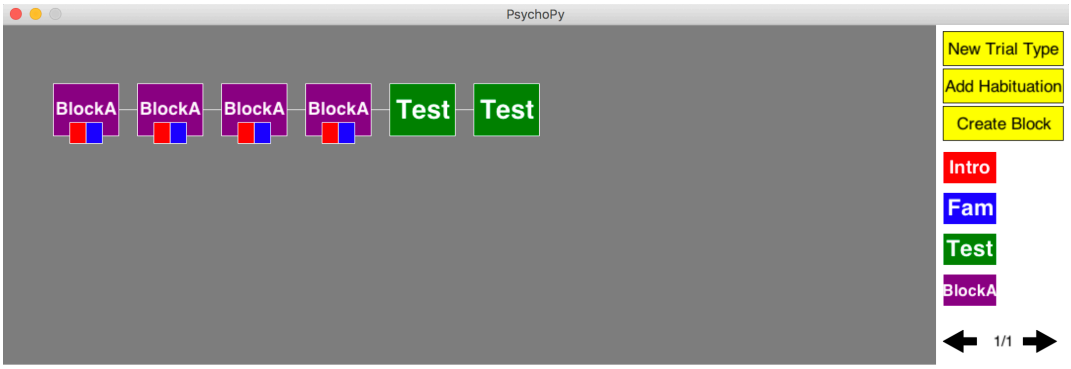


This is exactly like constructing the study flow, except that you can’t modify trial or block types. Click a trial type in the palette to add it to the block flow, click an entry in the flow to switch it with another entry in the flow, swap it with a different trial type from the palette, or remove it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

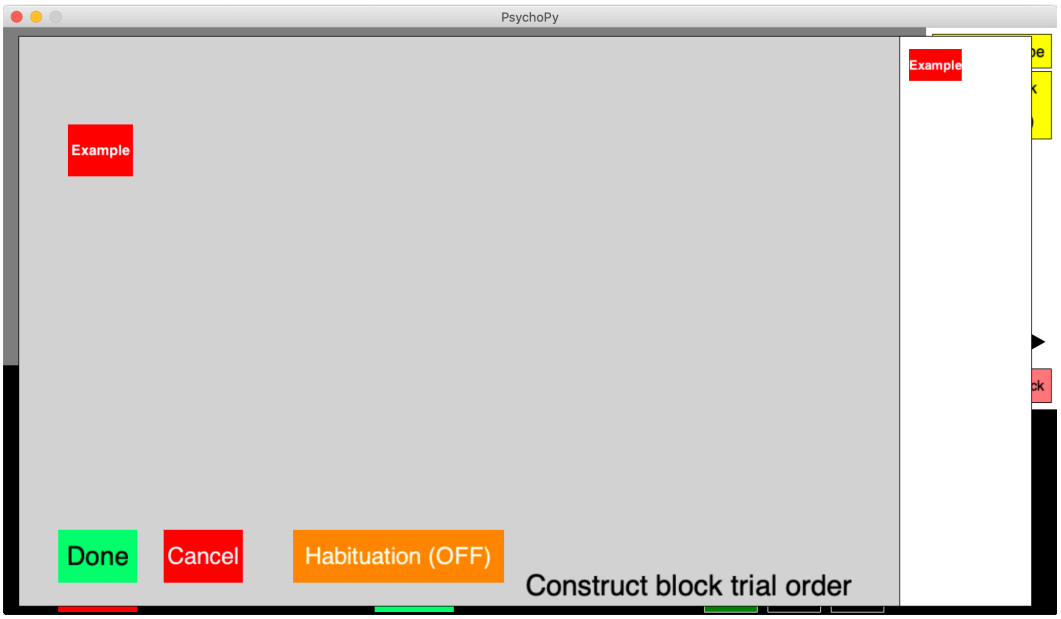
- **New in 0.10.1: Using "Block Redo", you can make it so that redos and aborts rewind to the start of a block iteration**. This is the default behavior for habituation blocks. If you have a block with three trials ABC, and you enable the "Block Redo" button found on the bottom of the block interface, then if you abort during trials A, B, or C, or redo after the end of one of those trials, the experiment will rewind to trial A, no matter where it's starting from. Basically this makes it so the "redo" and "abort" functions apply to the whole block rather than the trials within the block. Without this, by default the redo function will only pay attention to whether the trial is an auto-advance trial, and the abort function will only ever repeat the current trial.

Habituation

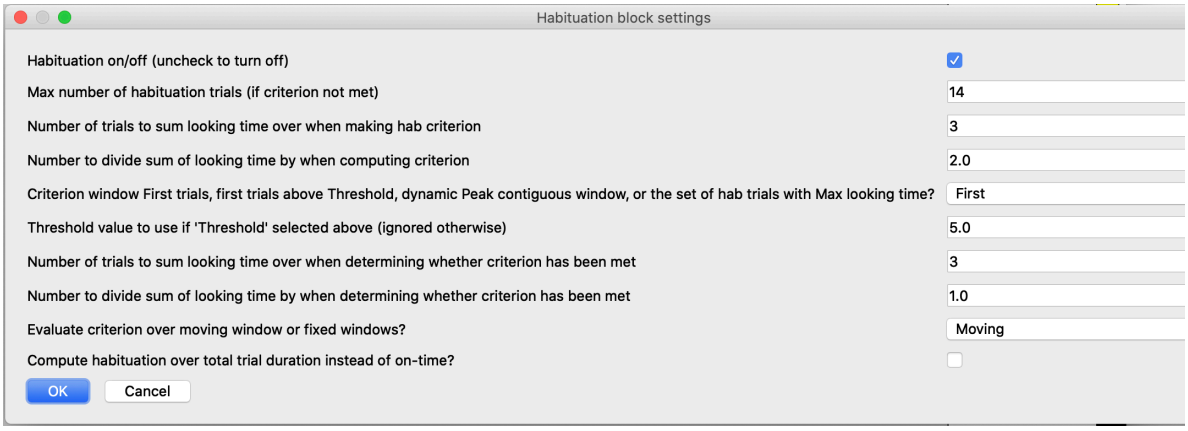
Habituation blocks or 'Hab' blocks are a special type of block (see Blocks). Habituation designs are studies where participants see a specific trial or set of trials repeated some number of times or until they have become habituated to them, that is, their looking-time decreases below a habituation 'criterion', determined by their looking time earlier in the block. Once the criterion has been met, the block ends and the experiment jumps to the first trial after the block. There are a couple of special considerations about habituation blocks:

1. **A given habituation block can only be added to the study flow once**. However, you can have multiple separate habituation blocks in an experiment. This is necessary so that
2. **Habituation blocks cannot be part of other blocks**. They are always the 'top-level' block.

To make a habituation block, click the "new block" button, then in the block-maker screen, click the "Habituation" button at the bottom to open the habituation settings.



The first item in the habituation settings is a check-box to indicate whether this block is a habituation block or not. If checked, this block will be used as a habituation block.



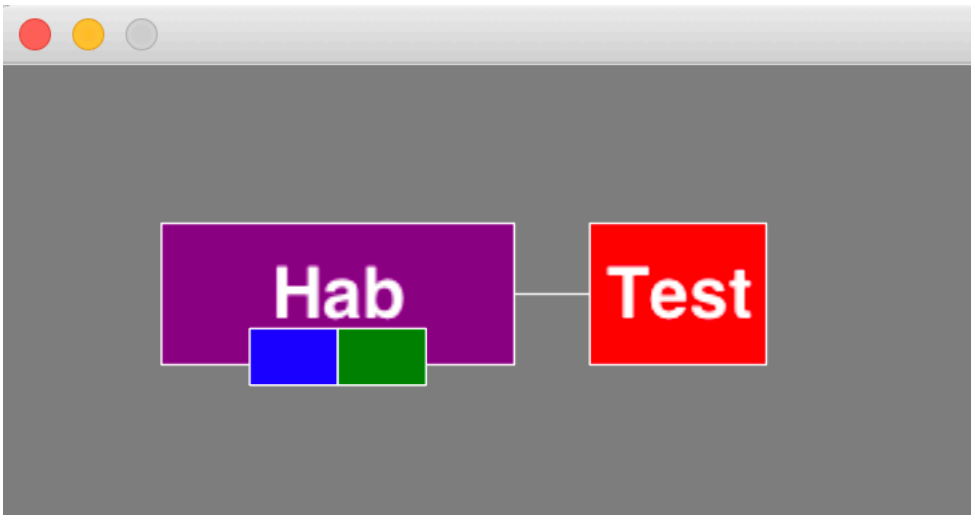
If your block has multiple trials, there will be check-boxes letting you specify which trials are counted for the habituation criteria, **both setting the criteria and determining when they have been met**. Let's say you had a habituation design where you wanted to show infants a movie followed by a 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. Then, in the block interface, you would open the habituation settings and there would be two

check-boxes at the bottom, one for each of the trials in the block. You would un-check the movie trial, and leave the image trial checked. In the experiment, all habituation calculations would only consider looking time to the still image, though the movie would still be presented normally.

If your experiment contains a habituation 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 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' shows up double-wide in the study flow because it's not just one instance of the habituation block, it's a variable-length habituation sequence, up to the maximum number of habituation 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’ block, followed by a test trial.

To modify the habituation settings for a particular block, just right-click the block in the palette and click the habituation settings button again.

Deleting trial and block types

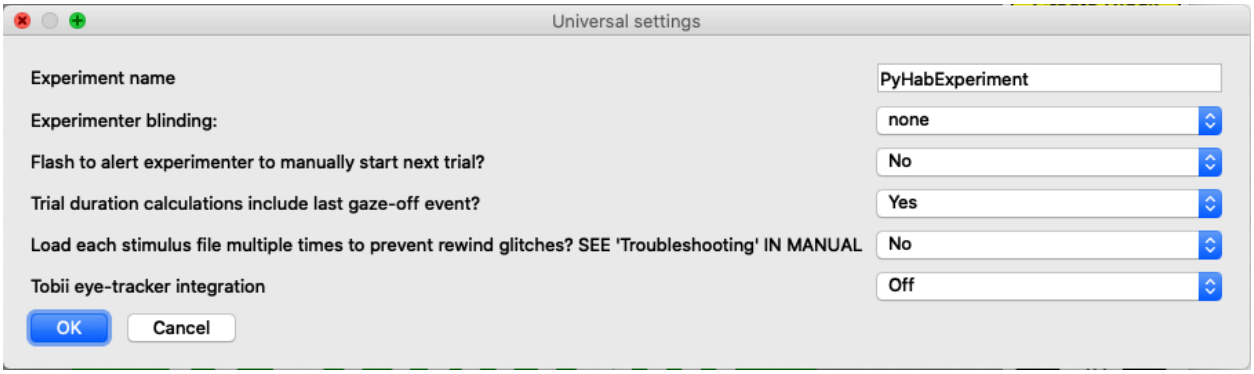
The “delete a trial type” button at the bottom of the trial palette is straightforward. Click it, and a dialog will open with a drop-down list of the trial and block types you have created. Select the one you wish to delete and hit “OK”. If you select a trial type, this will remove *everything* associated 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 third 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.

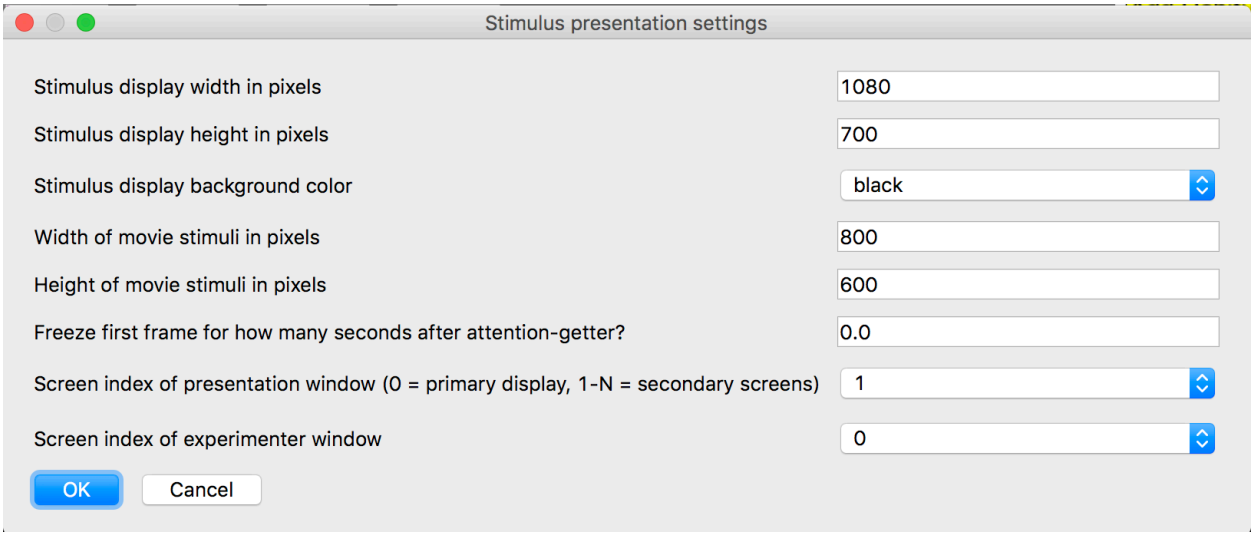
The fourth setting affects how trial durations are recorded in the data file and for purposes of determining habituation (if you set habituation to use total duration rather than on-time in the Habituation settings). By default, duration is just duration, start of trial to end of trial. However, if you select “No”, then if the infant was looking away at the end of the trial (e.g., if the trial ended *because* the infant was looking away), then the last gaze-off event will be subtracted from the duration. This option is included because different labs have expressed different preferences about how to record trial duration.

The next setting is a little dangerous and should only be used if you’re having trouble with your study displaying movie stimuli. See Movie stimuli freezing or playback issues.

The final setting is only relevant if you want to use a Tobii eye-tracker with your experiment. PyHab has the ability to interface with most Tobii eye-trackers without any additional customization. There are three options, “Off”, “Record only”, and “Control advancement”. “Record only” means that eye-tracking data will be recorded but will not be used in place of a human’s manual coding using key-presses. “Control advancement” means that the eye-tracker will be used *instead of* a human coder to determine when trials begin and end, and all

gaze time calculations will be based on whether the eye-tracker detects the infant looking at the screen or not. See Using PyHab with a Tobii eye-tracker for more details.

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.

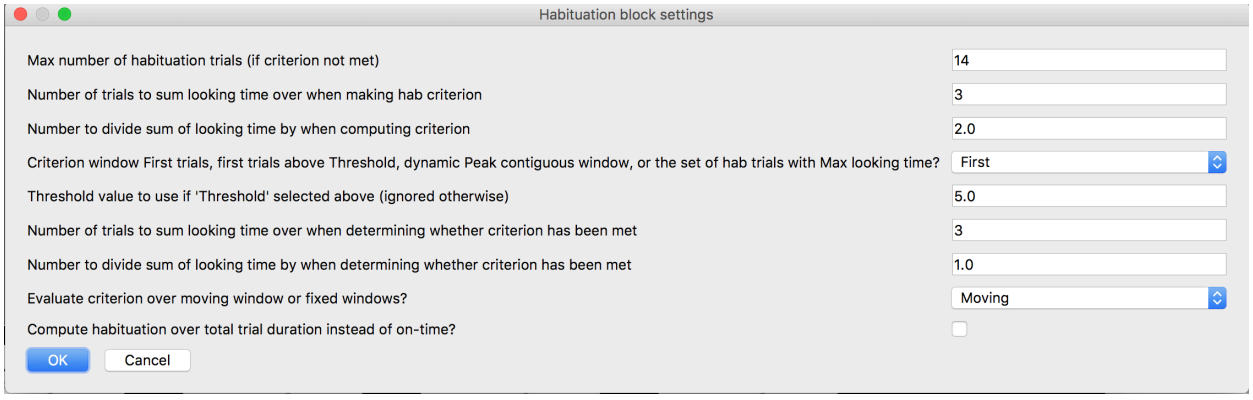
The next line imposes a minimum delay between the end of the attention-getter and the start of the stimulus. I recommend setting this to at least .1 if you are using PyHab to control stimulus presentation, because if there is no delay the sound from the attention-getter can conflict

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.

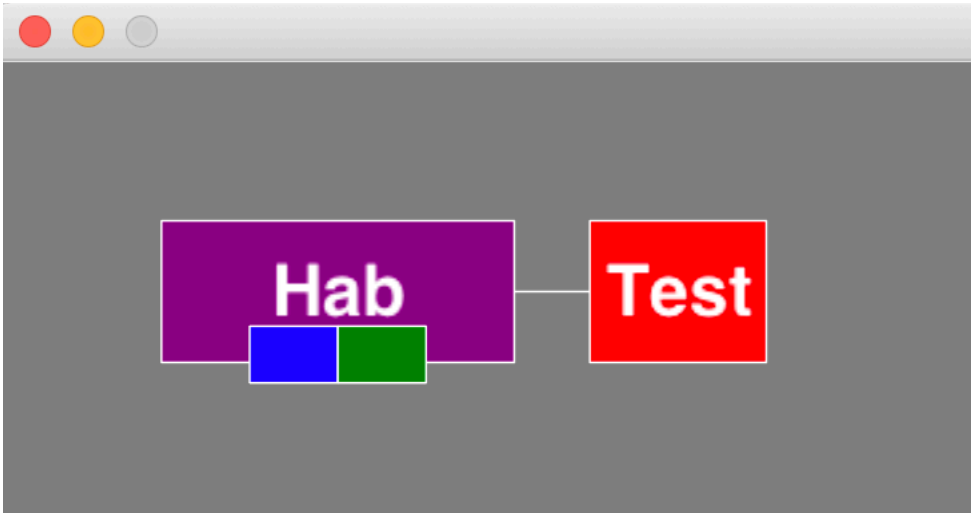
The "Screen index" fields are basically "which monitor should the stimuli appear on, and which monitor should the experimenter appear on". By default, on most computers, the "secondary monitor" will be screen index 1, while the primary display will be screen index 0. The drop-down menu will show you 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

Habituation settings are controlled at the level of the individual habituation block. You can get to the habituation settings by opening the block interface and clicking the 'Habituation' button. If you only had one trial in the block, it would look like this:



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 repetitions of the habituation block, the test trial will be shown, no matter what (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, but 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. Finally, the "Last" habituation type computes the habituation criterion based on the last N habituation trials where N is the "number of trials to sum over". This allows you to do things like compute the total over the last 5 trials, and then see if the last 1-2 trials account for less than some proportion of that total.

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.

Finally, the check-box at the very end allows you to do all habituation calculations on the basis of total trial duration rather than on-time. This is a straight toggle, if you check this box all calculations will be done over total trial duration, otherwise they will be done over on-time.

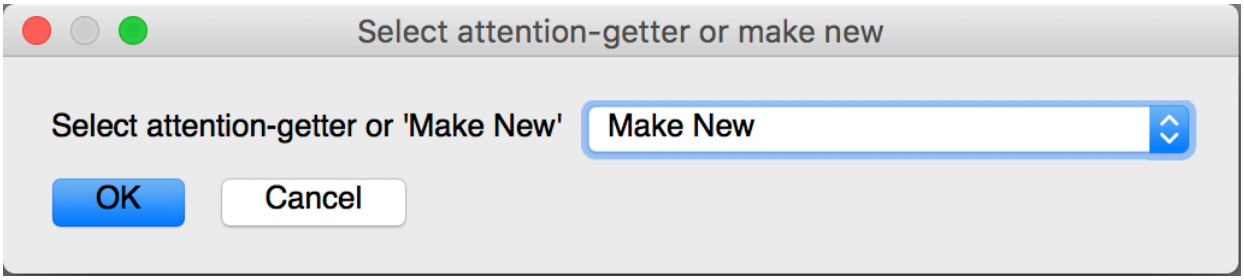
If your habituation block has multiple trials in it, under this check-box will be check-boxes for each of the trials in the block. If a trial is checked, that trial will be included in habituation calculations (both setting the criteria and whether it has been met). If unchecked, it will be ignored.

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

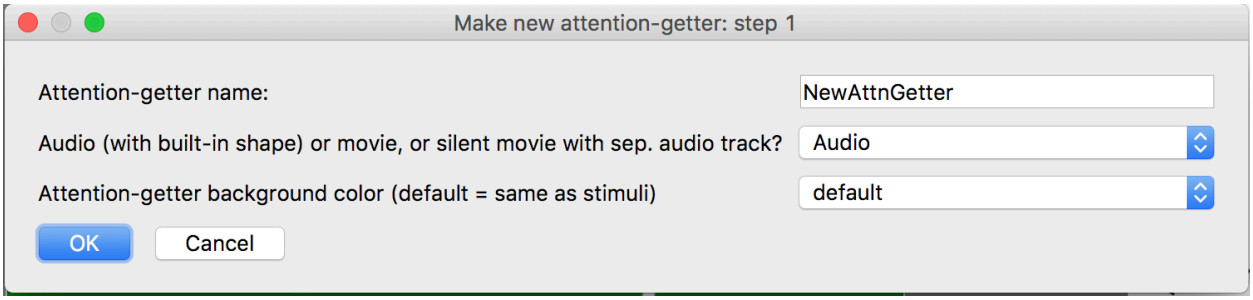
always available, which consists of a looming yellow rectangle and a rising musical scale. However, you can make your own, either a movie file, or an audio file with a procedurally-built shape.

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.

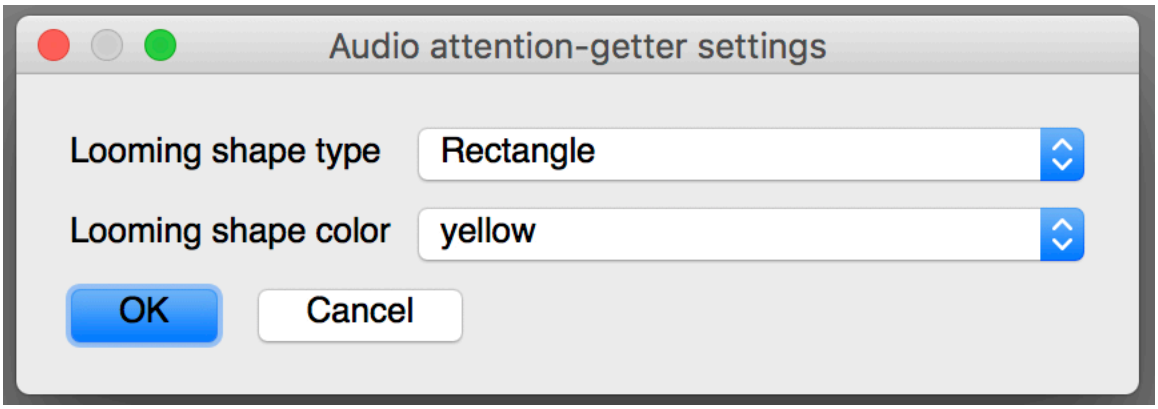
When you hit OK, the next window that pops up will look like this:



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. You can make one with audio and a geometric shape, use a movie file, or use a movie file along with an audio file. The third line lets you set the screen background color around the attention-getter, in case you want it to differ from the background color of the rest of the experiment (which is set in the Stimuli settings).

After hitting OK, you will be prompted to create your attention-getter. Movies are 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. If you choose Movie + audio, you will then be prompted to select an audio file 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.

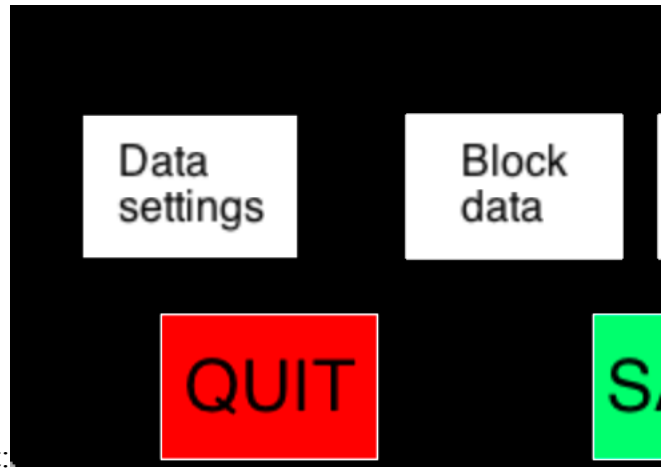
There are two extremely important things to know about these settings:

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 the type of your study (between single-target, preferential looking, and HPP), 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.

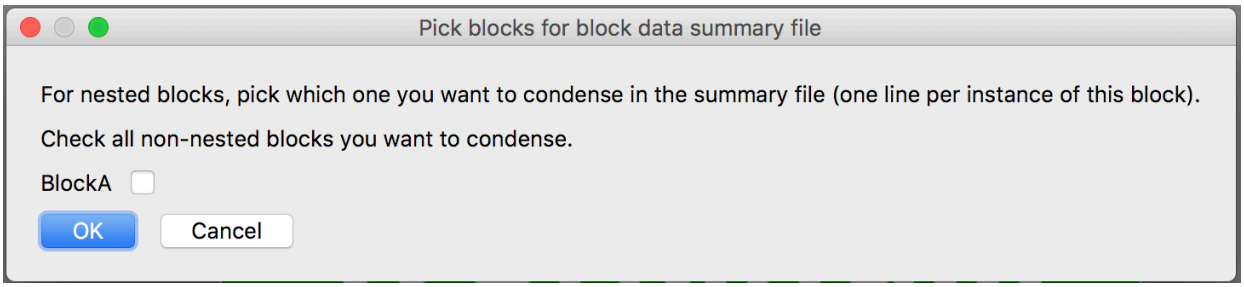
Block data settings

If you have created a block, the "data settings" button will change and a new button will appear next to it:



The "Block data" button does something incredibly simple: It allows you to create a block-level summary file. So, instead of one line per individual trial, it gives you one line per block. It otherwise obeys the data settings. So why do you need a settings window at all? Well, what if you have multiple blocks, and they're hierarchical, so some blocks appear in other blocks? Do you want one line for the top-level block or the low-level block? Also, what if there are some blocks you want condensed in this summary file, but other blocks you want one line for each trial in the block?

That's what this button controls, and it controls it in a very simple way. Click on the button, and a dialog will appear with a list of all of the valid block types that you can condense in the data file. Simply check the box for a given block, and in the summary file there will be one line per instance of that block, which will combine all of the on-time and off-time data from every trial in the block.

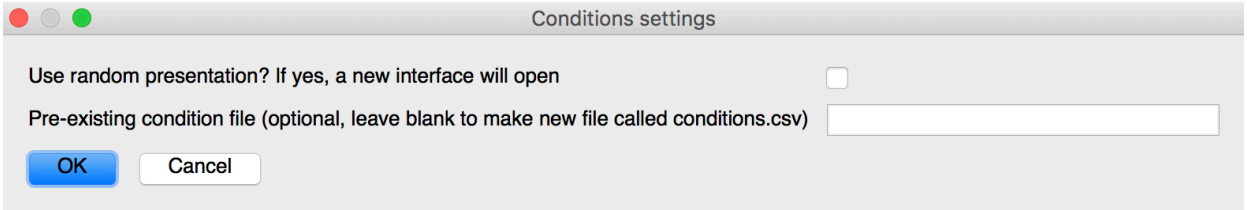


Conditions in PyHab are basically a way of overriding the default order of stimuli or order of trials within a block. This is extremely handy for counterbalancing or between-subjects manipulations. Each participant sees one condition, so conditions are a way of specifying the exact stimuli, and order of stimuli, that each participant will see.

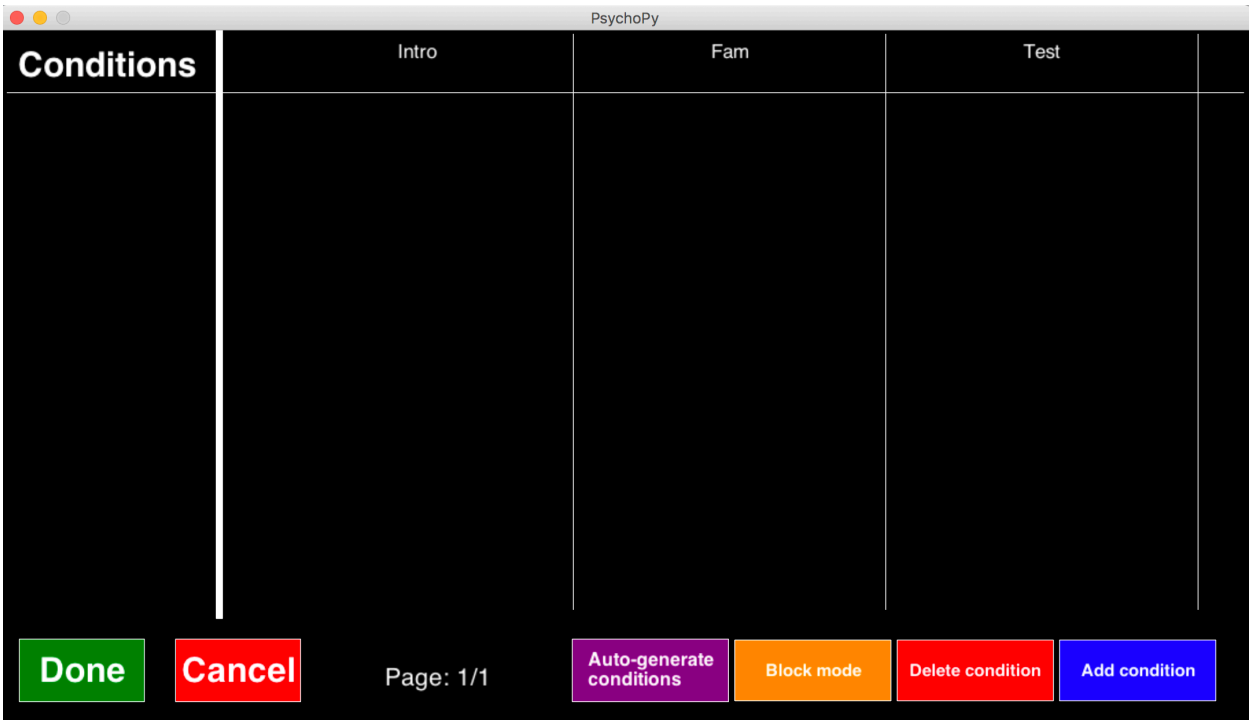
Here is where things get a little complicated. Before you start messing with these settings, here's what you need to know:

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. **Note: If there is a file in your experiment folder called conditions.csv and the box is unchecked, when you check the box, you will overwrite that file with the new conditions when you save.** 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:



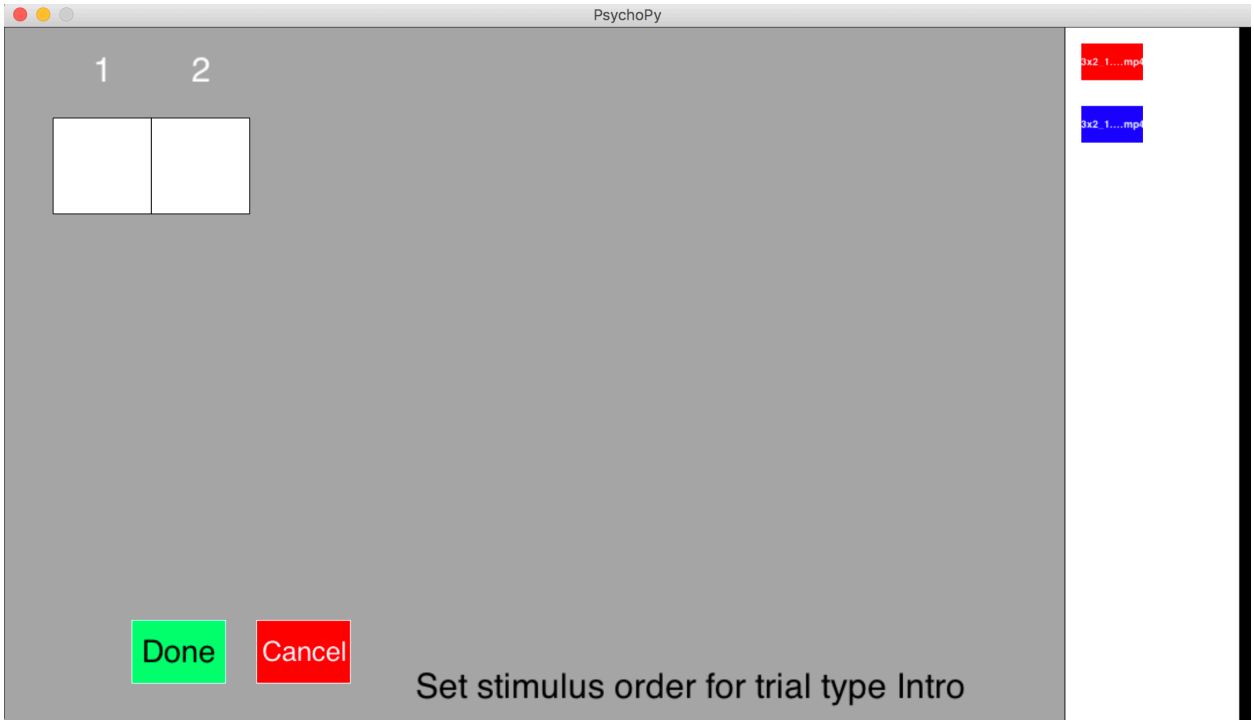
The "Done" button in the bottom-left saves your conditions and exits to the regular builder. The "Cancel" button just exits to the builder but doesn't save anything. Across the top you have the list of conditions in the left column, and then each trial type in another column. There are four other buttons 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). This button will only appear if you have blocks in your

experiment. The last button will say either “Auto-generate conditions” (if there are no conditions present) or “Randomize over subjects” (if there are conditions already, or you are in HPP mode). 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 (regrettably, it does not work for HPP experiments, which are too complex for this). “Randomize over subjects” allows you to create a blinded, counterbalanced random order of conditions.

Each function is explained below.

Add condition

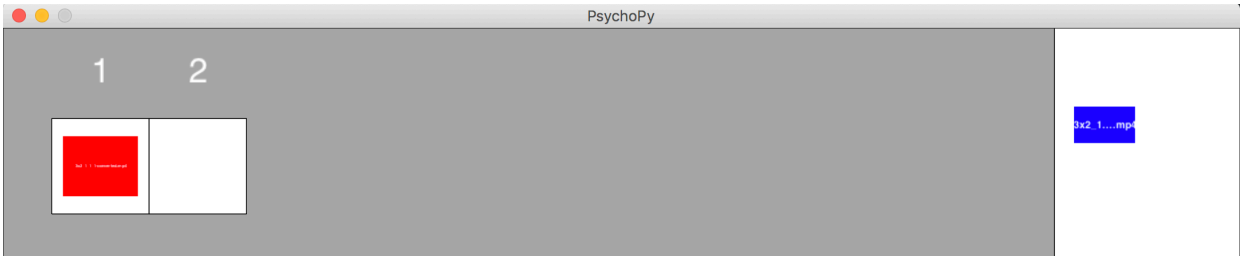
When you click on “add condition”, you will first be prompted to create a name for the new condition. You can name it whatever you want. When you hit “OK”, you will go through a series of screens that look like this:



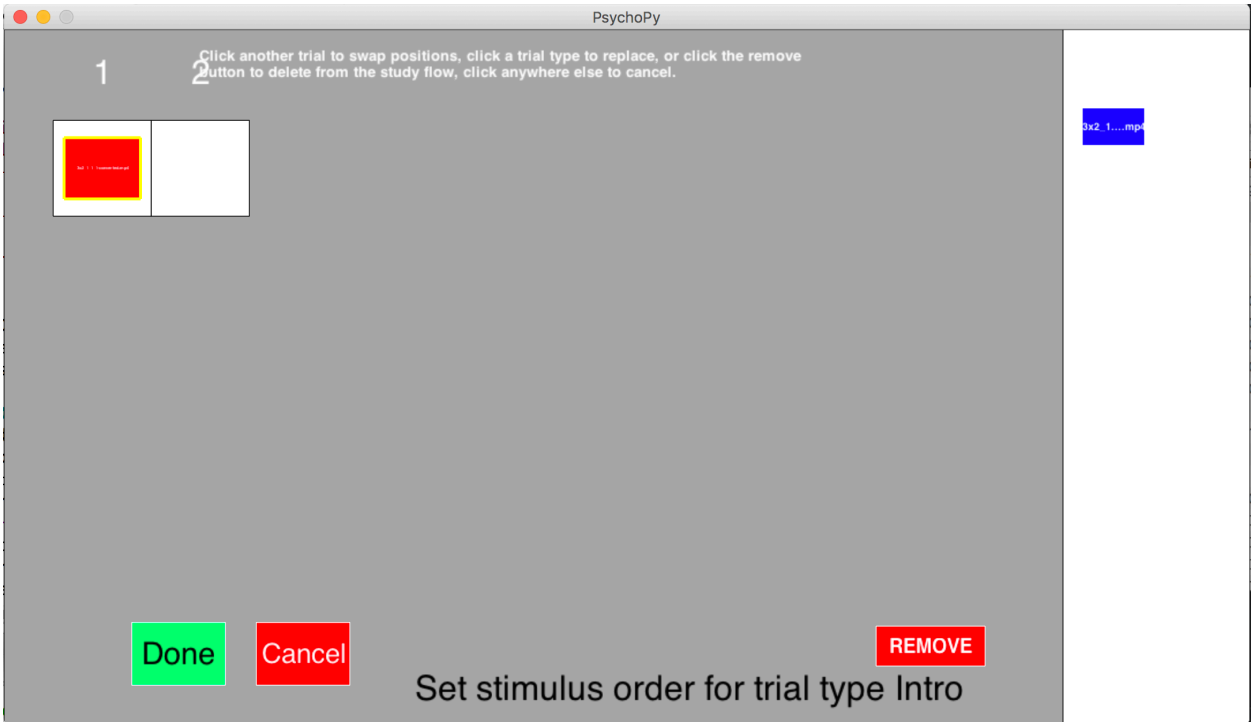
You will see one of these for each trial type in your experiment. In each one, what you are doing is setting the order of stimuli for that trial type in that condition. The stimuli assigned to the trial type are listed in the palette at the right (much like trial types are listed when constructing the study flow). The white numbered boxes represent iterations, i.e., the first time trial a trial of type Intro is presented in this condition, whatever is in Box 1 will be shown. The second time a trial of type Intro is presented in this condition, whatever is in Box 2 will appear. If there is a third instance of trial type Intro, it will loop back to one.

The number of available iterations will be equal to the number of stimuli assigned to that trial type. **You do not need to fill every iteration. You can make it so that certain stimuli never appear in a given condition by simply not adding them to the condition in this interface. For participants in that condition, it will be as if that movie does not exist.** Each trial type in each condition needs at least one stimulus assigned to it, but no more than that.

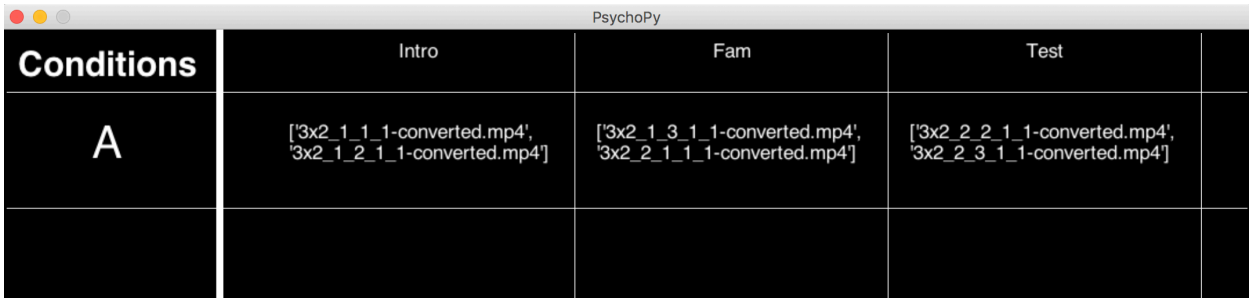
To add a stimulus file to the condition, simply click on it in the palette. This will remove it from the palette and put it in the first available iteration.



If you need to change the order of stimuli, just click on it in the flow and you will be able to move it or swap it the same way you move and swap trials in the study flow (I know, the text display is garbage here, it's on the list of things to fix).



Once you have finished adding stimuli to a trial type, hit "done" and you will go on to the next trial type, and so on until you have done this for every trial type. 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.

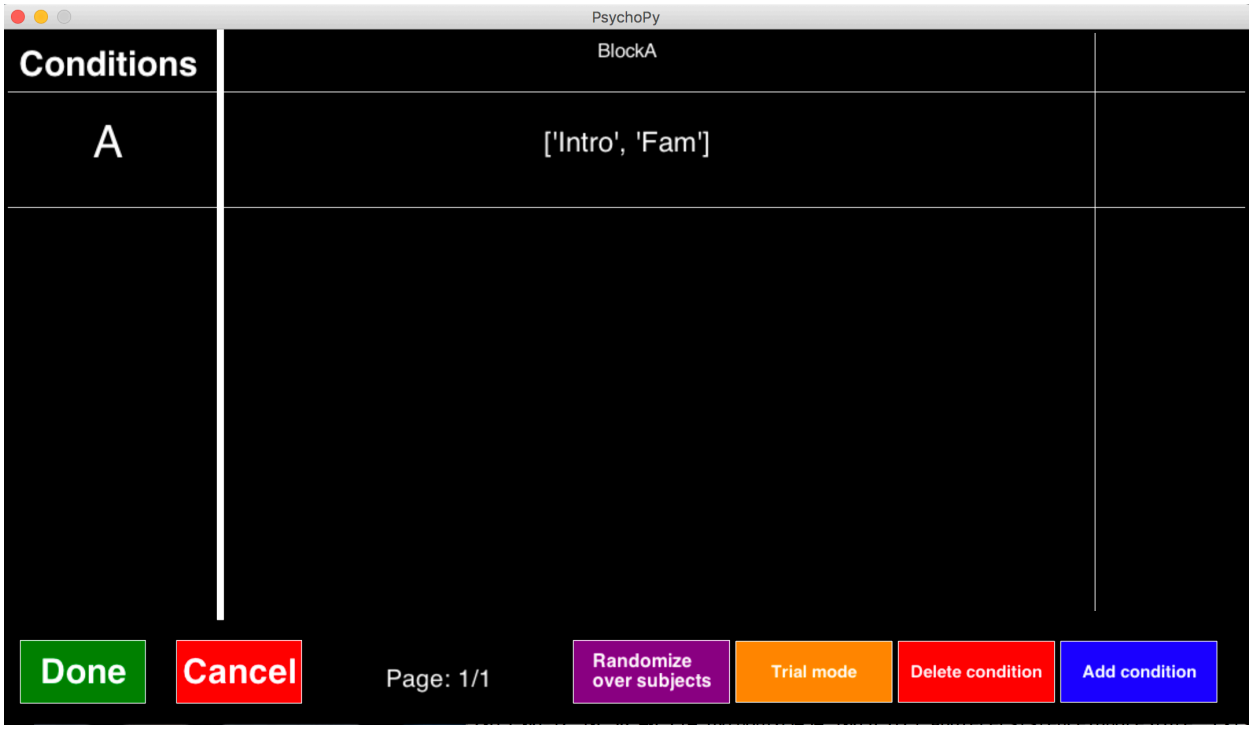
You can modify a condition by clicking on the row it occupies (this simply re-opens the same interface as "add condition", but with the condition's current order pre-set), and delete it using the "delete condition" button, which will bring up a list of conditions much like the "delete trial type" button. 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 so it varies between conditions. If you clicked “Block mode” after making the condition described above, you would see this:

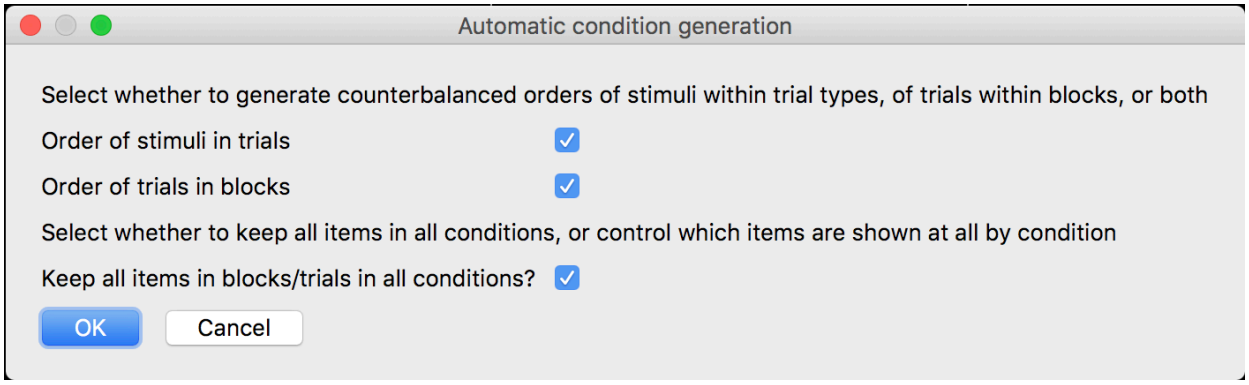


If you are in block mode and click “Add condition”, or click on an existing condition, you will again be prompted to name the condition, and then you will see an interface very similar to the one you used to create the block in the first place. **In this case, you are not changing which trial appears in each instance of the block in the trial flow, you are changing the order in which the trials will appear in every instance of the block.** Basically, you are re-creating the block for each 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.

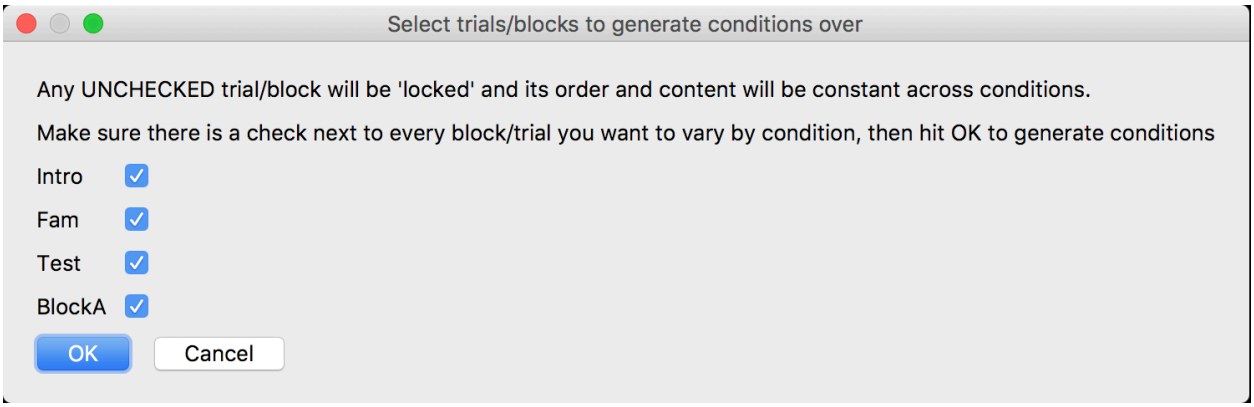
Auto-generate conditions

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):

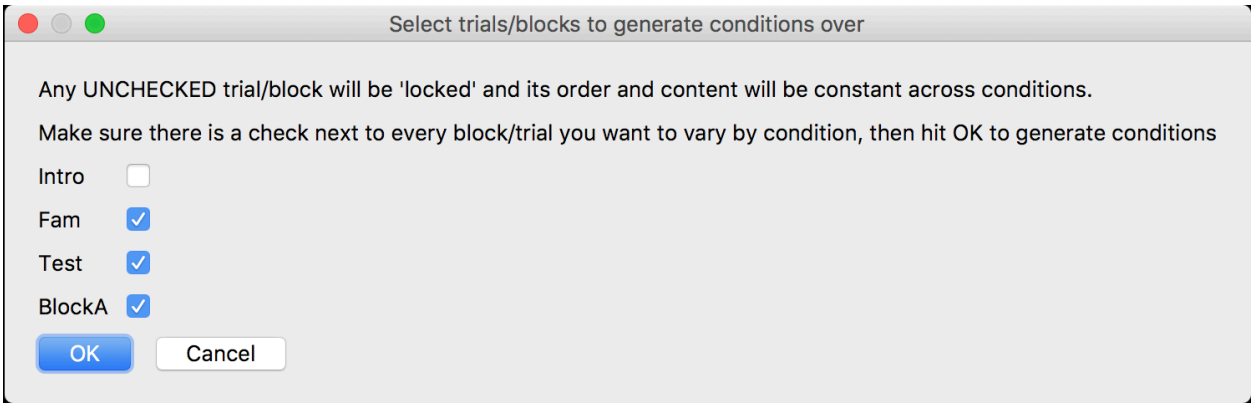


You can choose to generate conditions that counter-balance the order of stimuli in trials, the order of trials in blocks, or both. The last check-box controls whether the generated conditions ensure that every participant sees every trial and stimulus, or if they manipulate, between-subjects, which things 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).

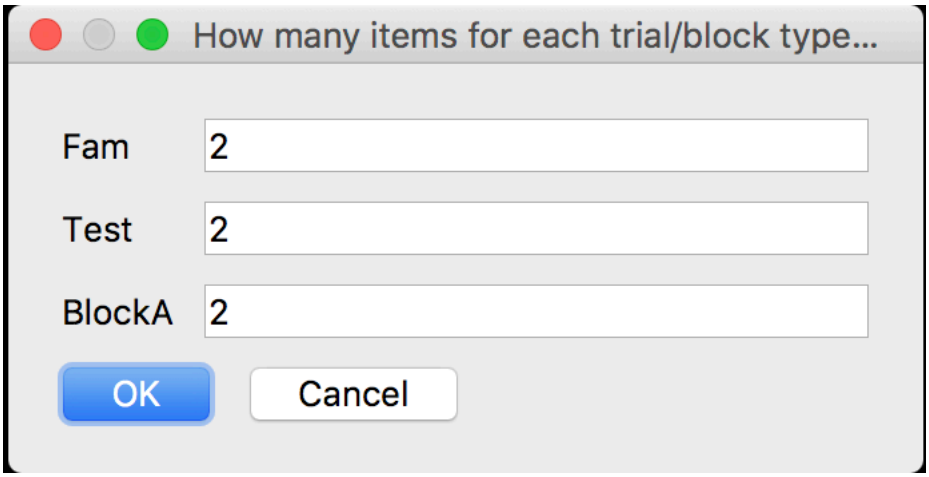
When you click "OK", the next dialog that appears will ask you if you want to "lock" any trials or blocks.



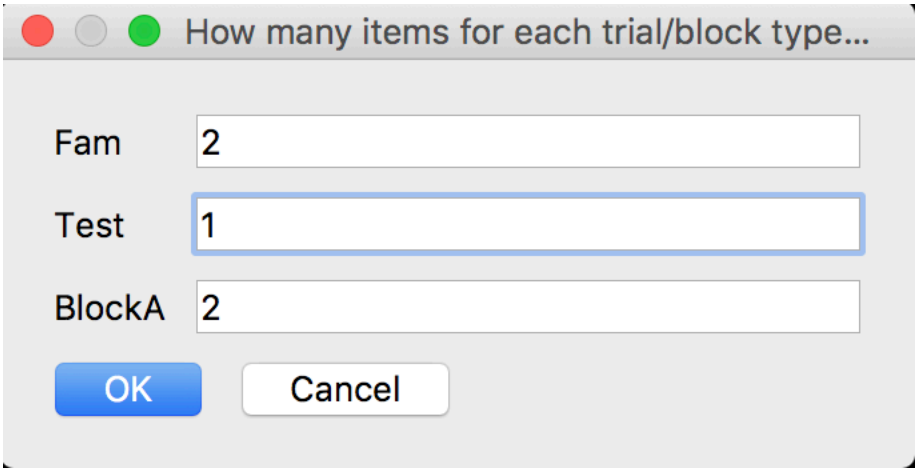
If you don't want to counter-balance the stimuli in a given condition or the order of trials in a given block, simply un-check it here, and items in that category will appear in their default order in every condition. So, for example, if we wanted the intro trials to always be the same, we would uncheck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

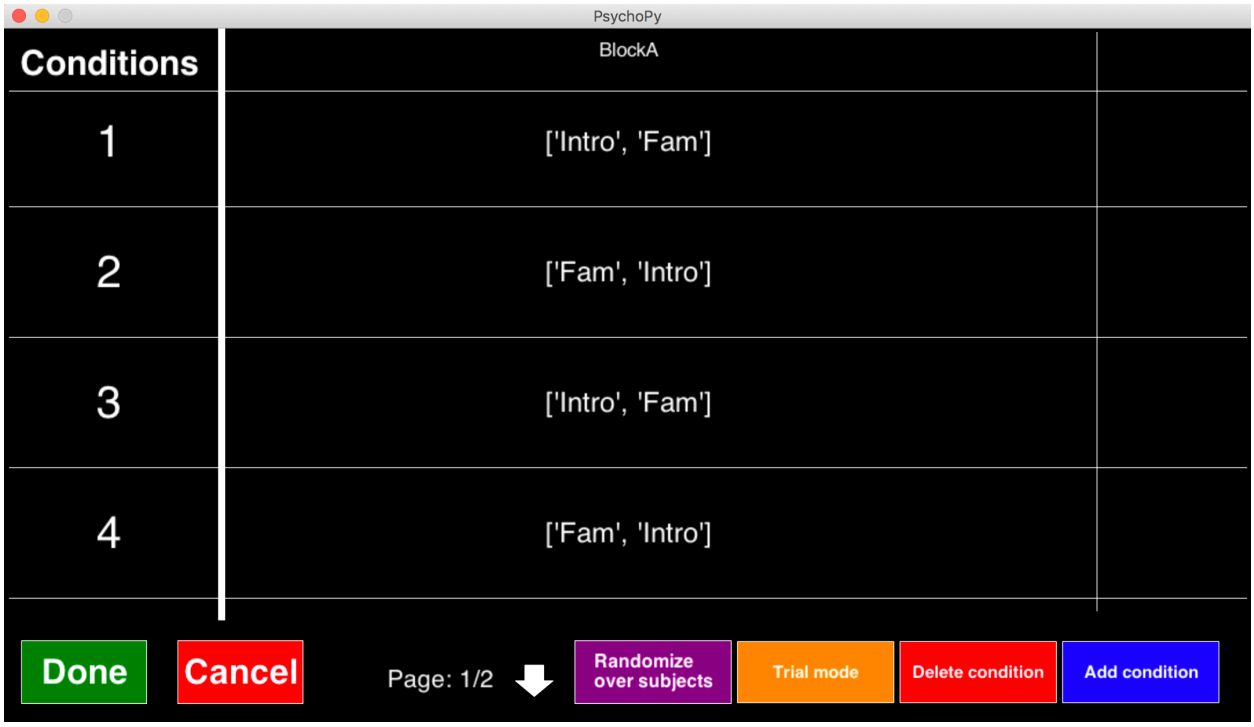


Note that the "locked" trial type, "Intro", doesn't appear here. Now say that we wanted to make it so that each participant only saw **one** stimulus during test trials. So, a between-subjects manipulation. We would simply set "Test" to only have one item per condition:



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

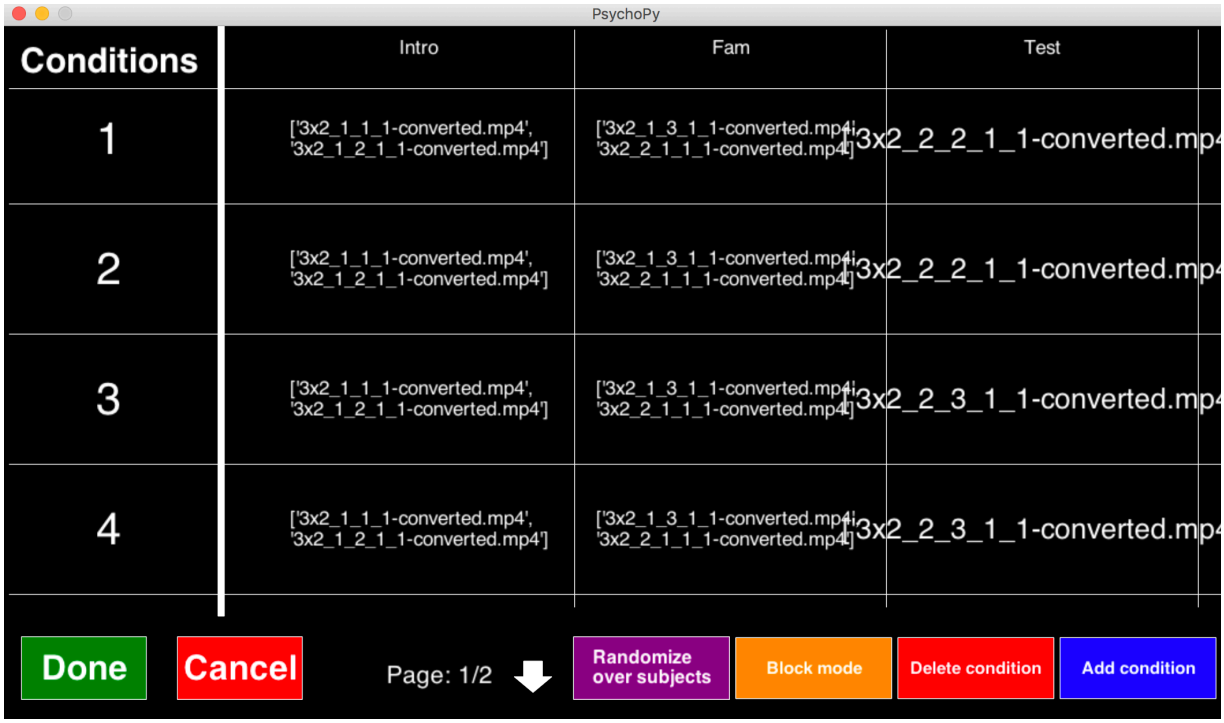
Note that there are a total of 8 different conditions across 2 pages. "But wait", you might be thinking, "aren't conditions 1 and 2 completely identical?" Not quite. If you click on "Block mode":



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.

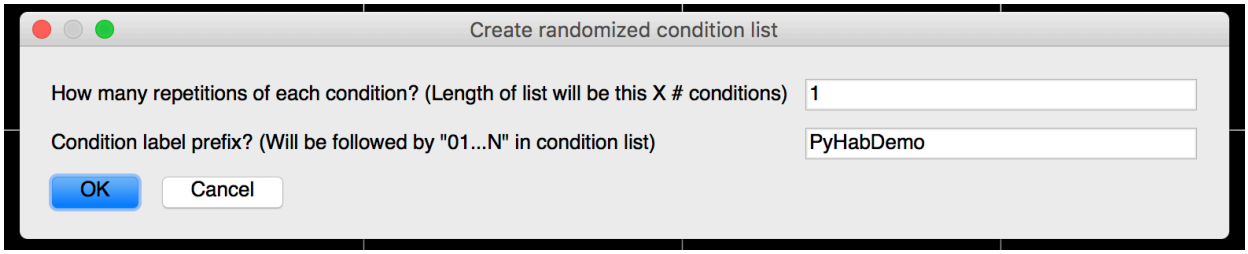
Creating randomized conditions by participant

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



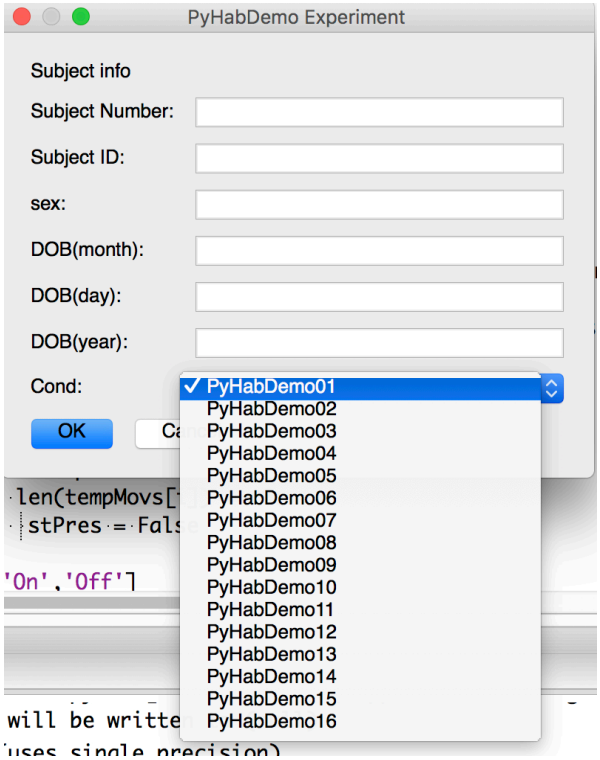
Say you plan to recruit 16 participants for your experiment, and you want to randomly but evenly assign them to each of these eight conditions. Furthermore, you want to be blind to the condition for each individual participant, so the person running the study doesn't know what condition they are currently 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.**

This introduces an important but subtle distinction of terminology: There’s “conditions” in the sense of “unique orders of stimuli/trials”, and “conditions” as they appear in the list after randomization. The former I call “base conditions”, the latter is just a way of ensuring that base conditions are randomly but evenly spread among participants.

Head-turn preference procedures

Head-turn preference procedure study designs are very different from other types of studies. For PyHab's purposes, the big difference is that PyHab has to juggle up to three different stimulus presentation screens as well as the experimenter display! So how can you control what appears on which screen and when?

When you change the builder to HPP mode, PyHab change a bunch of the settings menus to allow you to control what appears on which screen and everything else. It is definitely a little more complex than building a study with only one stimulus presentation screen. This section details how the PyHab builder works differently when making an HPP study. There are a few things to know up front:

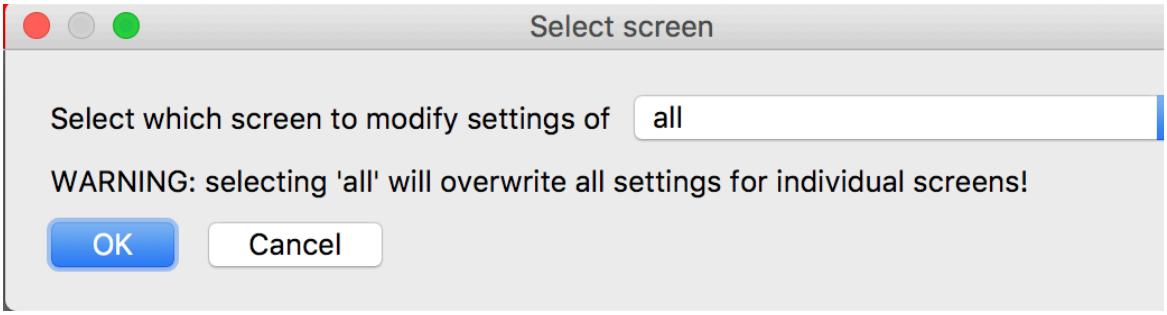
- **Set the experiment type to HPP as early as possible!** When you change a study to HPP mode from some other experiment type, PyHab has to update a bunch of stuff and erase a bunch of existing settings. You can save yourself the trouble of redoing the conditions, data, and stimuli settings and set the builder to HPP from the start.
- **PyHab can only control which screen visual stimuli appear on, it cannot directly control which speaker audio stimuli come from.** If you need to change whether something comes from the left speaker or the right speaker, your best bet is to change the stimulus file itself to only use the appropriate sound channel.
- **You will pretty much need to use conditions**. The condition system is basically the only tool in PyHab that is able to accommodate the possibility of presenting stimuli on multiple screens simultaneously and presenting stimuli in different orders. If you don't have any conditions, by default all stimuli will appear one at a time on the central screen, like a single-target study. If you want to do anything more complex than that, you will need to use conditions (but you probably would anyways).
- **Currently, the start/end screens and attention-getters will only appear on the center screen**. This will change in a future version of PyHab, you will ultimately be able to control which screen everything appears on. In the meantime, you can work around this

by creating trial types that basically serve as attention-getters, that show short movies on whichever screens you need them to appear.

- **If you don't have a center screen, don't worry!** The center screen's "window" always appears behind the left and right windows, and you can set it so nothing is ever drawn there. So, your stimuli will appear on the left and right screens unobstructed and you can basically ignore the existence of the extra window.
- **Advanced trial setting: Only count looks to screens w/stimuli as 'gaze-on' events.** This setting can be found in the "Advanced Trial Settings" menu in the builder. If this setting is checked for a given trial type, then for that trial type, all the 'gaze-on' events that control the study flow (i.e., when a trial begins and ends, as well as habituation calculations) will **only** count looks to a screen where stimuli are being presented. Any looks to a screen where there are no stimuli will be counted as a gaze-off. However, **this only affects study flow, all data will be recorded veridically**. Your data file will still show gaze-on events to non-stimulus screens as gaze-on events, but during the experiment it will behave as though they are gaze-off events.

HPP stimuli settings

In HPP mode, the function of the "Stimuli settings" button changes slightly. When you click on "stimuli settings", you will first see this dialog:



This lets you choose which screen you want to change the settings of: Center, Left, Right, or all of them. Once you select the screen, the regular stimuli settings dialog will appear, but all of the settings you make will only apply to the screen you selected.

One setting that is particularly important is the "screen index". For non-HPP experiments, you will usually only need to worry about indexes 0 and 1. For HPP, you will be juggling up to four different screen indexes (Left, Right, Center, and the experimenter display). The screen index setting controls which physical screen is left, right, or center. You may have to do some experimentation to figure out how your computer has mapped the indexes, because every setup is a little different about this. The good news is that once you have figured it out the first time, you should be set.

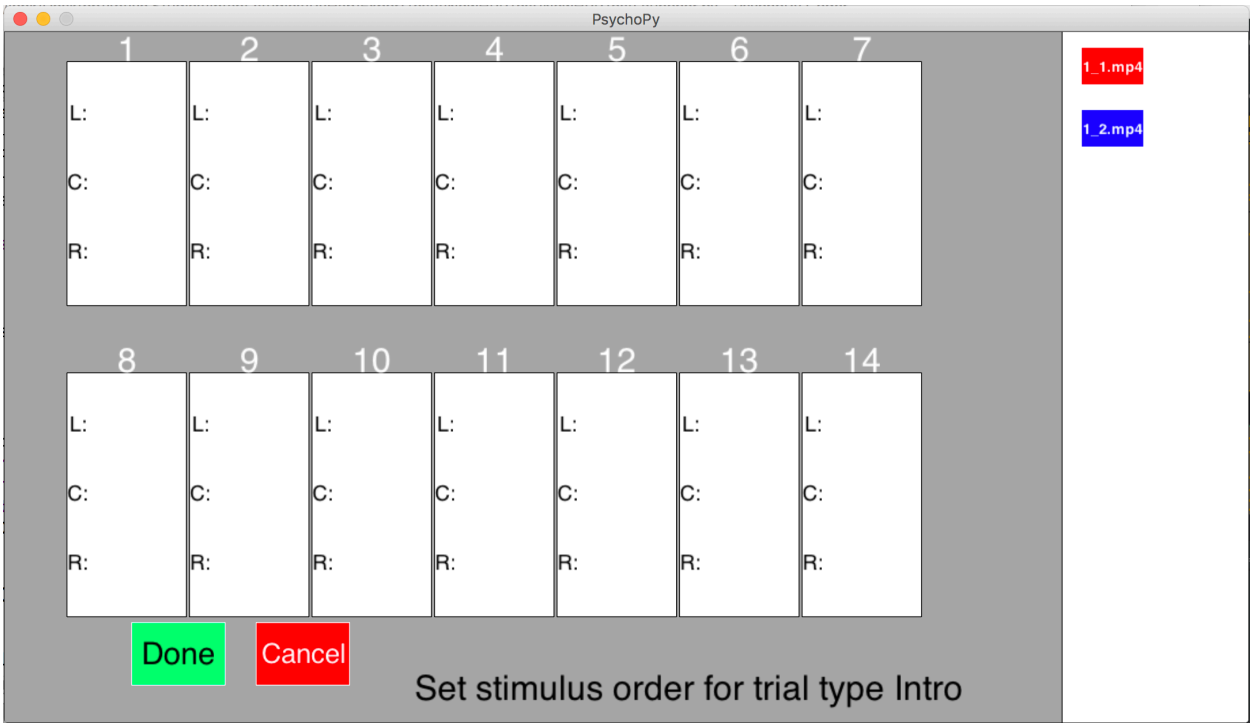
Screen indexing

Figuring out which screen has which screen index is not always simple or intuitive, so I made a little tool to help you figure out which screen has which screen index. In the PyHab download is a script called "ScreenIdentifier.py". Open this in the PsychoPy coder view and run it. It will pop up a small window on each screen with a number, which is the screen index of that screen. Press any key to close all windows when done. Using this tool you can figure out which screen index you should assign for your left/right/center screens and your coder screen.

HPP and conditions

Conditions are already one of the more complex piece of PyHab. Conditions for HPP studies are both essential and even more complex. In HPP mode, conditions are basically **how you control which screen stimuli appear on and when**. When you open condition settings in an HPP experiment, it will initially look very similar to how it does in a non-HPP experiment. When you click "add condition", you will once again be asked to set a name for the condition.

Hit OK, and a new interface will appear that looks like this:

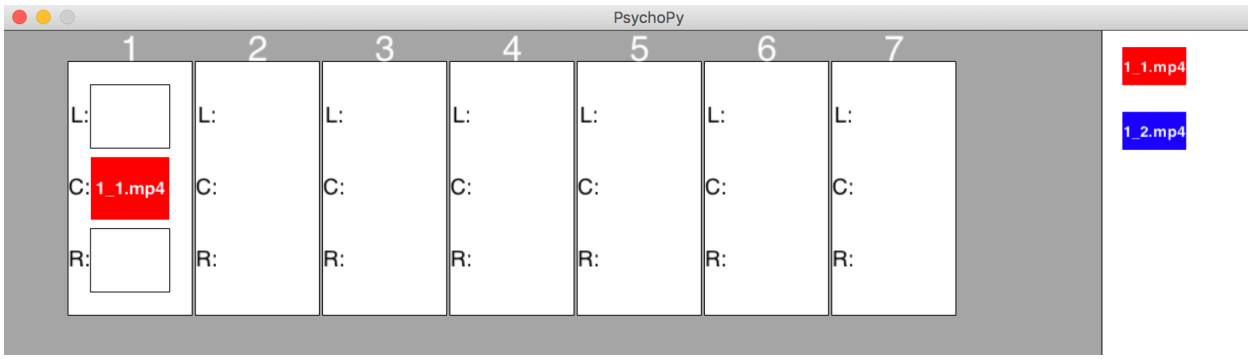


This interface is obviously different from the normal condition interface, but it shares many similarities. First of all, each white box in the flow is once again representing what will be presented in a single instance of that trial type. However, unlike the normal condition interface, you now have three slots in each. The letters in each box represent which screen the stimuli will appear on.

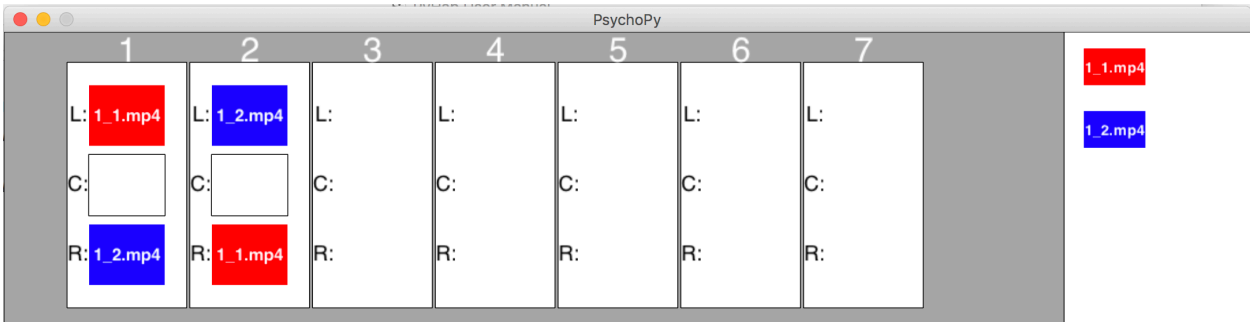
You can have stimuli appear on multiple screens at the same time (that's rather the point of HPP), and you can even have the same stimulus appear multiple times in one iteration of a trial or across multiple iterations of a trial. The number of iterations is not constrained by the number of stimuli, but only iterations with stimuli in them matter. Any iteration that is completely blank will be ignored.

In this interface, rather than just clicking on an item in the palette to add it to the condition, you drag and drop (click and hold down the mouse on the stimulus in the palette, and drag it to where you want it to appear). There is one major constraint: **You can only drag it to an open spot in the lowest-numbered unoccupied iteration, or an iteration that already has a stimulus in it.** For example, in the initial condition you see above, you could only drag one of the stimuli to one of the three slots in the box labeled "1". Let's see what happens when

you do that:



Note that the stimulus is not removed from the palette. Note also that two empty white boxes appear in iteration 1. Currently, this says that on the first instance of trial type "intro", the stimulus file 1_1.mp4 will be presented on the center screen, and nothing will be presented on the left or right screens. You could now drag a stimulus either to the left or right screens in iteration 1, or any box in iteration 2. Now let's look at a more complex ordering:

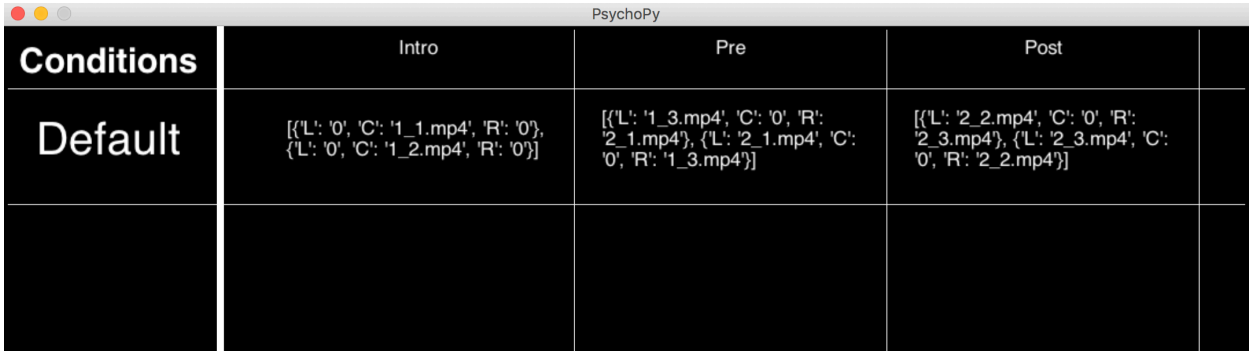


With these settings, on the first instance of trial type Intro, 1_1.mp4 will appear on the left screen and 1_2.mp4 will appear on the right screen. On the second instance of trial type Intro, 1_1.mp4 will appear on the right screen and 1_2.mp4 on the right screen. If there is a third instance of the trial type, it will loop back to the settings for the first iteration, and so on. You could now drag something into slot 3, if you wanted to set something different to appear on the third instance of this trial type. **Note: You could also have the same stimulus appear on multiple screens within the same iteration, e.g., have 1_1.mp4 appear on both the left and right screens in a single instance of the trial.**

To change something once it has been placed, just click on it as you would in the study flow interface, and you will as usual have the ability to either swap or remove it. If you remove the last stimulus in a numbered box, that iteration will be cleared completely. If that iteration is

not the last iteration, all subsequent iterations will automatically be moved back one. You cannot at any time have blank iterations (because the trial would attempt to display nothing).

When you have set the order and location of stimuli for a trial, hit “done”. When you have done this, you will return to the condition screen. The condition will be much more complicated than it is for non-HPP studies, but it will still have the order of trials. It will just now have what appears on each screen in those trials. Any screens that do not show stimuli on a given iteration of a trial will have “0” next to them.



Don’t worry if this seems complicated, it is! HPP experiments by their nature allow many more permutations than single-screen experiments. This is why you can’t auto-generate conditions for HPP experiments, it would be very difficult to specify how you wanted to constrain which permutations appeared.

Block mode for HPP experiments works exactly the same as it does for other experiments, because the order of trials within a block behaves the same no matter which screens stimuli appear on within each trial.

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.

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.

The PsychoPy 2020.1.0 and 2020.1.1 problem

Versions 2020.1.0 and 2020.1.1 of PsychoPy have a quirk that will break PyHab experiments made in versions prior to 0.9. The good news is that it is easily fixed. The short explanation is that these versions of PsychoPy (but not those before and probably not those after) change the "working directory" of the script when you run it. Whereas PyHab expects the working directory to be the experiment folder, 2020.1.0 and 2020.1.1 move it somewhere else. The solution is simply to tell PsychoPy to put it back in the same folder as the experiment launcher.

In order to do this, open your experiment's launcher file. On line 7 (the line after "import csv, os"), paste the following:

```
os.chdir(os.path.dirname(os.path.realpath(__file__)))
```

Save the launcher, and you're good to go. Experiments made in version 0.9 have this built in, and versions of PsychoPy from 2020.1.3 onward have changed this behavior back to the way it was before, but in this very narrow window, if you need to make a pre-0.9 experiment run in PsychoPy3 2020.1.0 or 2020.1.1, this is what you need.

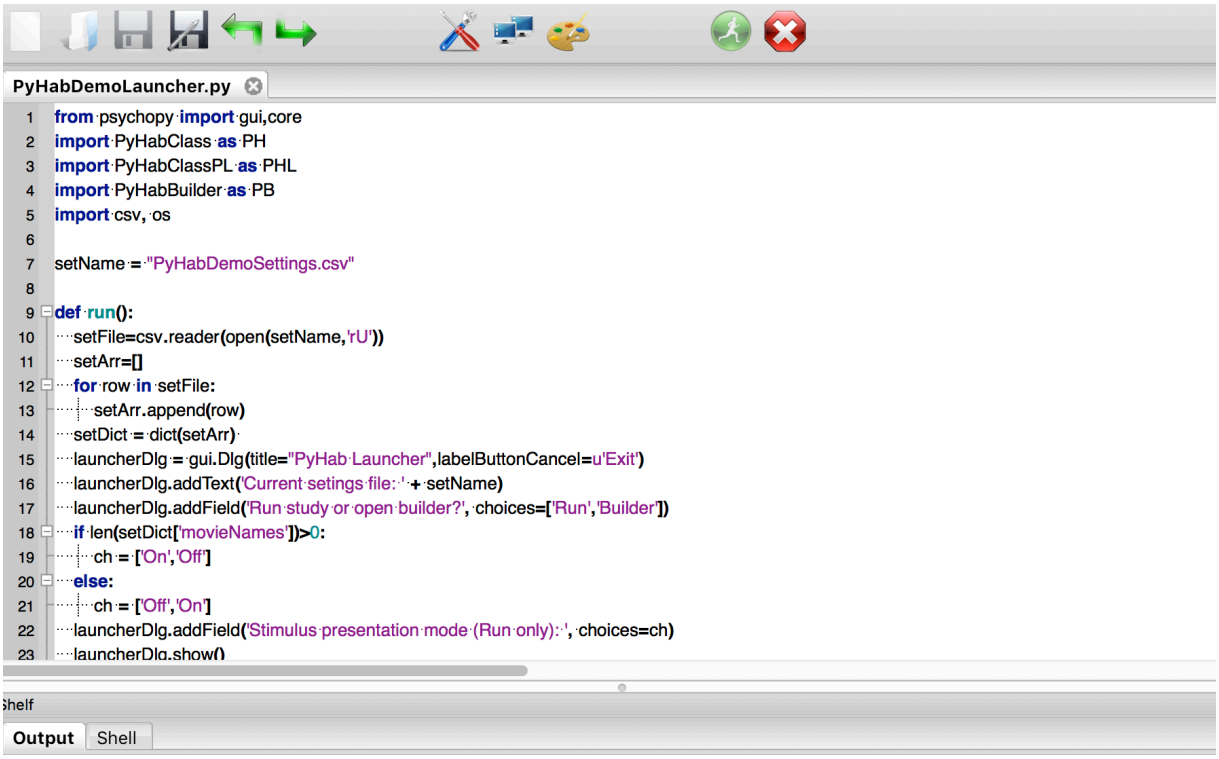
4. Running a study in PyHab

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.

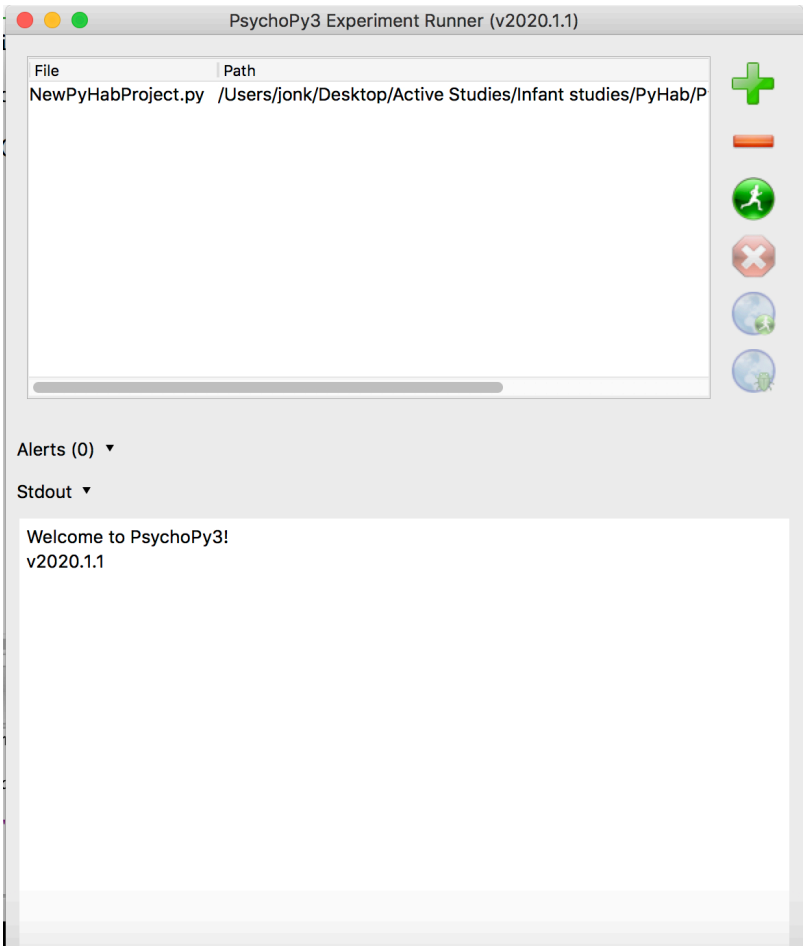
a. Starting a PyHab study

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.

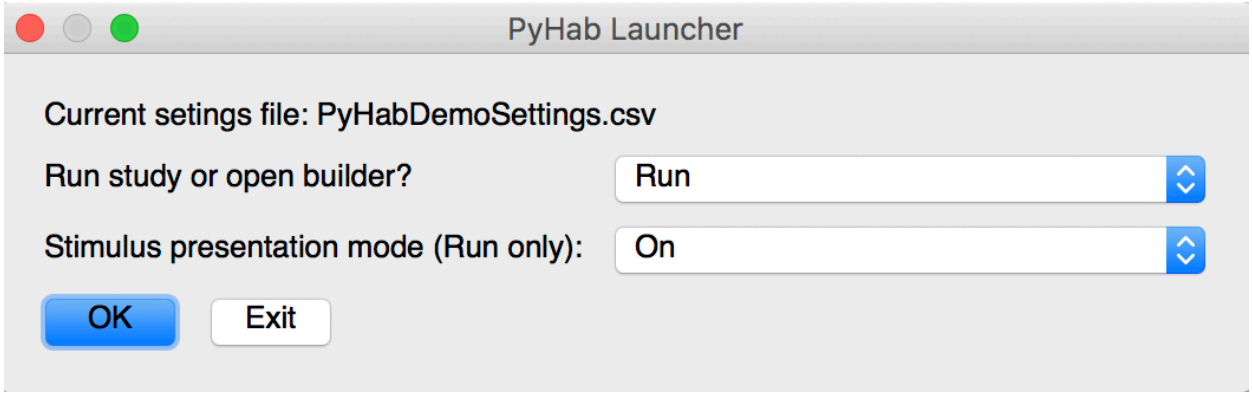
Once you have opened the script, your window will look something like this.



Simply click the green "Run" button in the top right. **In PsychoPy 2020.1.0 and later,** a new window will open that looks like this. Simply hit the green running man button again.

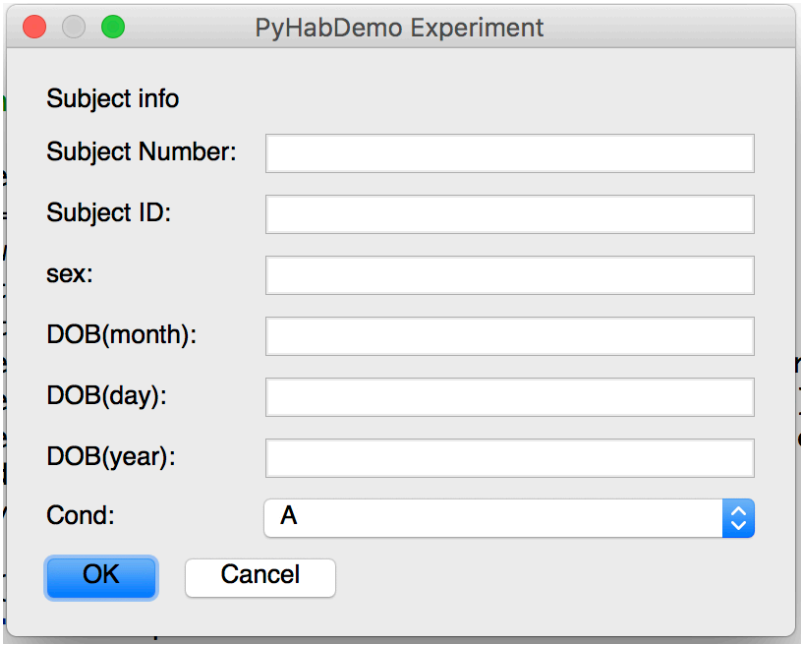


A new window will open that looks like this.



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:

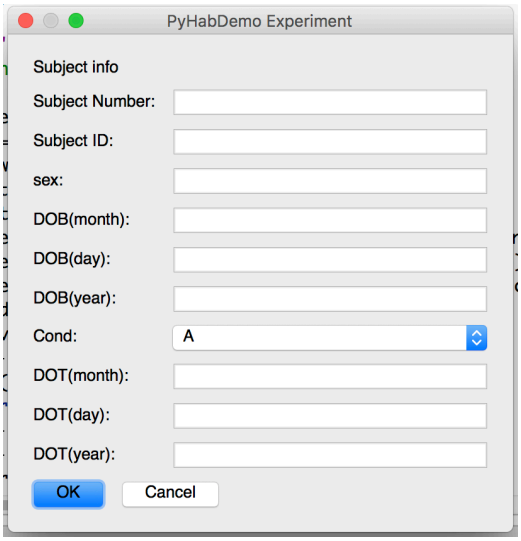


Here's what each of those fields does:

- 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.

[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.

- 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.**

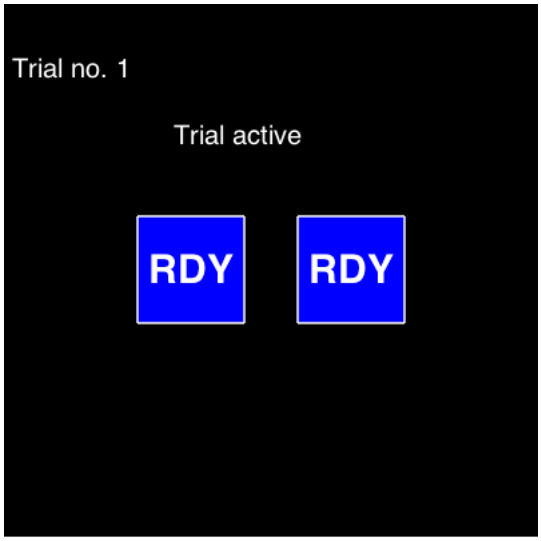


When you've entered all this information, hit OK. This will start the study run.

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 preferential looking, or gaze-on center screen for HPP). 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.
K	Skip attention-getter. Press during an attention-getter to immediately end it and start the trial.
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.

V	Head-turn preference procedure (HPP) only. Gaze-on left screen.
N	Head-turn preference procedure (HPP) only. Gaze-on right screen.
C	Tobii eye-tracker integration only: Re-run calibration and validation. Can only be used between trials (i.e., when the start of the next trial must be manually triggered by pressing “A” or a gaze-on event).

c. Data and computing reliability

Data

PyHab will save between one and six .csv files at the conclusion of a study. All are saved to the experiment's "data" folder, or to sub-folders within it. In the main data folder you will find:

- 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.
- trialDuration: The total trial duration. Note that the calculation can be changed in the Universal settings.

- 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).
- Eye-tracker .tsv data file: If you are using a Tobii eye-tracker with PyHab, it will save a .tsv file with the same name as the main summary data file. This file will contain all of the fixation data from the Tobii eye-tracker, as well as events for the start and end of each attention-getter and trial. It's formatted like any other .tsv file that you get from a Tobii eye-tracker.

In the "verbose" sub-folder, PyHab saves the following:

- 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, which is saved in the main data folder (not the verbose data folder). 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.

Finally, in the "timing" sub-folder, PyHab saves one additional file if the experiment was run in stimulus presentation mode:

- Timing: Since all of the time-stamps in the other data files are relative to the start of the trial, PyHab also saves a separate timing file that records the start and end of each trial and each attention-getter relative to the time the “A” key is first pressed to start the experiment. This file is also saved when running online studies. It’s a very simple file with four columns:
    - trialNum: The trial number, same as in the summary and verbose data files.
    - trialType: The trial identifier, same as in the summary and verbose data files.
    - event: Four possible values, “startAttnGetter”, “endAttnGetter”, “startTrial”, and “endTrial”
    - time: The timestamp of that event, relative to the beginning of the first trial.

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 as any other type of study in PyHab.

e. Head-turn preference procedure

Similarly, PyHab has a variant for HPP study designs. Much like preferential looking studies, this changes which keys you use, and the data that are recorded.

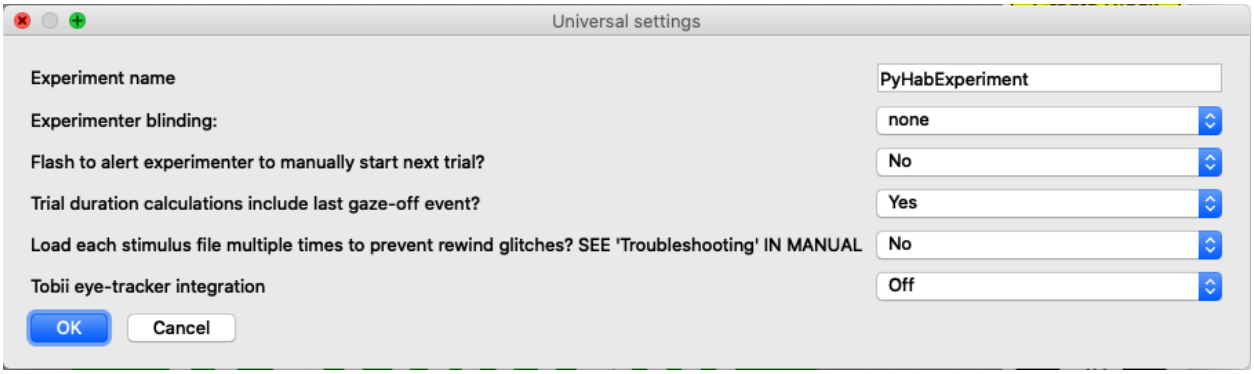
- Like preferential looking, HPP mode only supports one coder. Same problem, too many keys on the keyboard. You can still have one person code offline, however.
- The 'B' key is now "gaze-on Center". The 'V' key is "gaze-on Left" and the 'N' key is "gaze-on Right". They work the same except that it specifies which screen an infant is looking at. Any key can be used to start a trial, and time spent looking away (for determining when to end a trial) is only counted when none of these three keys are held down.
    - Note: You may need to be careful with these keys if you are having stimuli pause when the infant does not look at them, because this is set up to pause *on a per-screen basis*. In other words, if you set a trial so that stimuli pause when the infant is not looking at them in the advanced trial settings, then if the infant is not looking at the left screen (i.e. the 'V' key is not held down), the stimuli *on the left screen* will pause. More accurately, if you use this option, each screen will only play when the corresponding key is held down!
- The experimenter window, when running a study, will have three boxes rather than two, corresponding to the three screens.
- In the data, the looking-time columns will now be "sumOnL", "numOnL", "sumOnC", "numOnC", "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 screen, if it says R it refers to gazes to the right screen, and if it says C it refers to gazes to the center screen.
- The verbose data file now has four codes in the gazeOnOff column: 0 = "off", 1 = "gaze-on Left", 2 = "gaze-onCenter", 3 = "gaze-on Right".

Otherwise, running an HPP study is just like running other study in PyHab.

f. Using PyHab with a Tobii eye-tracker

PyHab, as of version 0.10.4, has the ability to interface with Tobii eye-trackers without any additional modification. This is mostly due to the existence of the "psychopy_tobii_infant" library created by Yu-Han Luo (https://github.com/yh-luo/psychopy_tobii_infant/). This library, which is included with PyHab, allows PyHab to send event timing to the eye-tracker and record fixation locations. **At the moment it is only usable with single-target studies, not preferential looking and not HPP.** The current implementation is fairly basic but functional. Future versions may add features.

To make a PyHab experiment that interfaces with a Tobii eye-tracker, you only need to change one setting, found in the Universal Settings panel in the builder:



The last setting in this window controls whether PyHab attempts to interface with a Tobii eye-tracker. There are three modes:

- Off: PyHab runs normally.
- Record only: When you start the experiment, before the coder window even appears, PyHab will attempt to connect to the eye-tracker and run a calibration and validation sequence. (More on calibration below.) During the experiment, all of the participants' fixations will be recorded to a .tsv file saved in the data folder, but trial advancement (when trials begin and end, habituation, etc.) is still controlled manually by the experimenter holding down a key to indicate when the infant is looking and releasing the key when they are not.

- Control advancement: As “Record only”, except that the eye-tracker will also be used **instead of** the experimenter’s manual coding to determine when trials begin and end, when habituation criteria are met, etc. All on-time and off-time data that PyHab records will similarly be determined by the eye-tracker. In both cases, this is done on the basis of whether the eye-tracker detects the infant looking at the screen at all, it doesn’t matter where on the screen they are looking. Fixation locations will be saved to the .tsv file as above.

Controlling calibration and validation:

The calibration and validation sequence is a little different from the rest of how PyHab works. The keys are different and the logic is different, largely because it’s not being controlled by PyHab at all, but rather by the pyhab_tobii_infant library.

In short, if you run an experiment in stimulus presentation mode, after filling in the participant information and pressing “OK”, the stimulus presentation screen will appear in black. To begin calibration, press the “1” key, and then when the infant is looking at the stimulus, press the spacebar. The system will lock up for a moment, this is unavoidable, but then the screen should revert to black. Then press the “2” key and repeat the process, and then for “3”, “4”, and “5”. After the last one, if the screen does not automatically advance, press the spacebar one more time and a calibration results screen will appear showing the estimated stability of each calibration location. If any of the points need to be recalibrated press the number corresponding to it and then press spacebar, or if they all look good then just press spacebar.

PyHab will then advance to a screen with five white dots, and a red dot representing the infants’ fixation location. You can’t exactly tell infants to look at each dot, but you should find that as they look around the screen, the red dot tends to go to the white dots rather than hovering in the middle of nowhere. If there is no red dot, the eye-tracker can’t tell where the infant is looking (i.e., it’s as if they’re looking away).

At this point, if you are satisfied with the calibration and ready to begin the experiment, press spacebar again and the normal coder window will appear, and you can begin the experiment as normal. If not, you can press the ‘R’ key and it will restart the calibration sequence.

The calibration and validation sequence can be re-done between trials by pressing the "C" key. However, this is only possible to use if the experiment has trials that are triggered manually (i.e., not all auto-advancing trials). It cannot be done during a trial or attention-getter.

5. Online studies

a. Basics and warnings

During the COVID-19 pandemic, mostly out of desperation, I created a modified version of PyHab that can be used to run habituation studies over the internet using a website called Slides.com (with which I have no affiliation or relationship) to control stimulus presentation while you record gaze behavior over Zoom or other videoconferencing systems. As of 0.10.4, this online version has been added to the main PyHab download, but it is different from the rest of PyHab in several ways. You can't just take an existing PyHab experiment and turn it into an online experiment. You will need to install some extra pieces into PsychoPy itself, get an account at a website that I have no commercial or professional relationship with and cannot vouch for in any way (but it's only being used to show stimuli, not record data, so it's not a privacy risk), and the timing is going to be a little noisy no matter what just due to lag.

In short, you should only use this system if both of the following are true:

1. **You need to run a gaze-contingent study online**. If your study's advancement isn't controlled by the infants' gaze behavior, and doesn't need to be actively monitored or controlled by an experiment, then LookIt (https://lookit.mit.edu/) is almost certainly a better solution. LookIt studies are run "unmoderated", meaning that parents do the study from home with no experimenter present. This has upsides and downsides, but the biggest limitation is that you can only code looking behavior after the fact. If you are doing something like habituation, you can't do it on LookIt, and that's where PyHab's online version comes in.
2. **Your stimuli require higher-quality video than Zoom screen-sharing provides**. It's certainly possible to run a normal PyHab or Habit study over Zoom using screen-sharing, though it introduces some annoyances. The main issue is that screen-sharing usually means your stimuli have reduced quality, because they are being loaded on your own computer and streamed in a compressed form to the participant's computer. If you have

animations that require smooth movement, the Slides.com presentation system PyHab gets around this problem. You direct participating parents to a website, and the website loads the stimuli directly in their web browser, meaning that they play roughly as well as they would play if you just sent them the video files themselves, so the quality should be the same on your computer and theirs.

Conceptual background

It may be helpful to understand how this system works in conceptual terms, because just thinking it as "PyHab but in a web browser" isn't exactly the right mechanistic model and might lead you astray. First, we need to talk about what Slides.com is and why it's related to this.

Basically, during the pandemic, I realized that there wasn't a good way to simultaneously present high-quality video and manually control the flow of an experiment over the internet in a "live" study. Most online studies with adults (and all LookIt studies) are "unmoderated", meaning they are fully automatic: no experimenter present, no appointment needed. Essentially, using systems like Pavlovia or Qualtrics or LookIt, the participant is downloading the experiment onto their own computer, running it, and then uploading the data. All of this happens seamlessly in the web browser, of course, but when you boil it down, that's what an unmoderated study is. That means it's possible to get fairly precise timing and present high-quality stimuli without any problems, because the only difference from an in-lab study is that the participant is doing it at home on their own computer instead of coming into the lab and doing it on a lab-owned computer. It means the experimenter doesn't have control over the computer's installed hardware or software, or the testing environment where the participant is doing the study, but in many ways it's not that different from an in-lab study.

Conversely, if the experimenter needs to be present for a study, one solution people often use is screen-sharing of some kind. During the pandemic we've all become familiar with this: you can simply show someone else on a video call whatever is on your screen. The downside of screen-sharing is that the experimenter is basically sending two video streams from their computer, their camera and a recording of their screen. Even if the experimenter turns off their own camera, whatever they send over screen-sharing isn't being directly rendered on the participant's computer, it's being rendered on the experimenter's computer and sent as a

compressed, low-framerate video. On the other hand, the experimenter can directly control which stimulus is shown when during the experiment, which isn't possible in unmoderated studies.

I wanted something where I, as the experimenter, could control advancement through the study, but the stimuli were being downloaded and rendered directly on the participant's computer. After a bit of trial and error, I stumbled across Slides.com. The basic idea is that a Slides.com presentation is a series of simple web-pages. The presenter (experimenter) can send the audience a link to a specific presentation, and the presentation is loaded in the audience's web browser. But, the key feature was that when the presenter goes to a new slide, **it also advances the audience's web browser to that slide**. The slide is still a website being loaded on the participant's computer, so it's still downloading everything and rendering it locally instead of compressing and streaming it, but **when each page is loaded** is under the presenter's control. In other words, it has the upside of an unmoderated study in that all the stimuli are being loaded directly onto the participant's computer, but like a moderated study, allows an experimenter to control advancement through the study completely.

From there I just had to figure out how to get PyHab to control a Slides.com presentation, but Slides.com isn't really set up for automation. In fact, the website only responds to key-presses and mouse-clicks, and there isn't a true API for presenting. The "simplest" solution was to have PyHab open a web browser that could be programmatically controlled to simulate mouse-clicks and key-presses, log in to a Slides.com account, navigate to the presenter URL, and then tell the web browser to navigate to each individual slide to present the stimuli for each trial. From a technical standpoint this is somewhat analogous to creating a lawn-mower by duct-taping a chainsaw to a go-kart. It is neither elegant nor a good idea, but in the absence of better solutions, it does work.

And again, I have no affiliation with Slides.com, and I don't think they know anything about this (nor do I want them to). It just happened to be the only existing solution for the problem I was trying to solve. Also, everything PyHab needs can be done with a free account, so at least as of now you don't need to pay to use it.

b. Extra installation steps

First, you must install Selenium into PsychoPy. In order to control a web browser, PyHab needs something that PsychoPy doesn't usually come with, which is a library called "Selenium". If you are one of those more technical people who have a Python environment in which PsychoPy is simply a library and you've been running your PyHab experiments from that environment, you can just use "pip install selenium". If you are using the PsychoPy app, it's a little more complicated:

1. Download the Selenium folder as a compressed .tar file from https://pypi.org/project/selenium/#files
2. Download the correct driver for your web browser. Go here and scroll down to 'drivers' for more information: https://pypi.org/project/selenium/#description
3. Unzip both files that you just downloaded (the driver may not need to be unzipped, it may just be a file on its own, but the other one will unzip into a folder called "selenium-N.NNN.NN" where the Ns are the version number).
4. **On Mac:**
    - In a new Finder window, go to "Applications", find PsychoPy, control-click or right-click, and select "Show package contents".
    - Go to contents/resources/lib/python3.6
    - In the unzipped selenium-N.NNN.NN folder, find the folder that is **just** called "selenium" and copy it into the python3.6 folder.
    - Now use the "Go" menu at the top of the screen and select "Go to a folder".
    - In the window that opens, enter "/usr/local/bin"
    - Copy the contents of the driver for your web browser into the usr/local/bin folder (e.g., for Chrome it should just be called "chromedriver"). It will probably ask you for an administrator password.
5. **On Windows:**
    - Go to Program Files\PsychoPy\Lib\site-packages

- In the unzipped selenium-N.NNN.NN folder, find the folder that is **just** called "selenium" and copy it into the site-packages folder.
- Go to Windows\System32
- Copy the contents of the browser driver (e.g., "chromedriver.exe") into this folder. You will need administrator permissions to do this.

Then, create a Slides.com account. You MUST use a username and a text-based password. DO NOT use oauth2, "Log in with Google", or anything like that.

c. Building an online experiment

Before you even open PyHab, you need to create your Slides.com presentation.You will want to set it up as follows:

- One slide for each stimulus item, e.g., each video.
- You must have each video set to auto-play (see below). Otherwise the parent will need to manually play the video.
- One slide for each attention-getter, which will need to also be a video file. The shape-and-audio attention-getters won't work here.
- One blank slide, or a slide with whatever you want families to see before starting the experiment and after it ends.
- The order doesn't particularly matter, but **make a note of the slide number for each stimulus item, attention-getter, and the blank slide**, because you will need those numbers later.
- When you're done making the slide-show, go to "presentation mode". A panel should open up on the left side of there screen, and under the heading "Present live" you should see a URL. Copy this, you will need it shortly.

Important slide setup tips:

- In the left menu, go to the little paintbrush icon ("style"), scroll down, and ensure that "transition" and "background transition" are set to "none". By default, the slide-show will "move" from one slide to another left-to-right as you go through the order, but because we'll be jumping around from slide to slide, this is a bad idea.
- Adding a movie to a slide is easy, just drag and drop it in. Check the box that says "autoplay" and the one that says "loop".
- **The slide numbering counts from 0**. The first slide is 0, the second slide is slide 1, etc. Keep this in mind when noting down the slide numbers!
- When you run PyHab in the lab, unless a trial is set to auto-advance, the movie won't start playing until the first gaze-on event. **In an online study, the movie starts playing**

as soon as the attention-getter ends. That means that if you want a still-frame at the start of a trial, you need to make an extra slide that is just that still-frame as an image, and you will add it as a separate trial type (see the end of this section).

Now that you have all of your slides made and all of your slides written down, it's time to create the PyHabOnline experiment. In PsychoPy's coder view, go to "Open", and then in the PyHab folder, look for the folder that says "PyHabOnlineEdition" (it'll be next to PyHabDemo most likely). In this folder, open the file called "NewPyHabOnlineProject.py", then run it. **Note that this is different from the NewPyHabProject.py file you use for in-person PyHab experiments.**

This will open up the builder view like normal. The way trial settings, blocks, and habituation work are all pretty much the same as normal PyHab, though they are based on an older version so some features are missing. There's also one big difference in how movies start that means you may need to add an extra trial type (see below). There are key differences in Stimuli Settings and how the stimulus library works.

- Open the stimulus settings in the builder and you will see that it now has no information about screen or stimulus dimensions. Instead, it asks you to fill in your login for Slides.com, i.e., the username and password.
    - Yes, this is necessary. Yes, it will be stored in plain text in the PyHab settings file. It's not being sent to anyone and it's only found in the settings file for the experiment, but don't use a password that you use for anything else and be mindful of who you share the experiment with.
    - This isn't a data security issue because Slides.com isn't recording any of the data, it's *just* presenting the stimuli. PyHab will record everything on the experimenter's computer and there alone.
- The stimulus settings also has a field for the presentation URL. Remember the presentation URL you copied from Slides.com earlier? This is where it goes.

- Finally, there is a field for the slide number of the blank slide. This is important because it's your start/end-of-experiment screen, the first thing parents see when they open the slide-show.
- The stimulus library works completely differently. Rather than loading the stimulus files themselves, when you go to "Add stimuli to library", you will be prompted to put in a name for the stimulus and a slide number. That's it. Once you've added a stimulus to the library, you can add it to different trial types like normal.
- The attention-getter settings also work slightly differently. You add a slide number and a duration. The duration is in seconds and you need to make sure it's accurate, because it tells PyHab when to end the attention-getter and proceed to the trial.
    - Err on the side of putting in a duration that's too short, because if it's too long the attention-getter might start the next loop and then suddenly cut it off to start the next trial which can look/sound weird.
- Other notes:
    - You can't make HPP online studies (there's no point, most participants don't have three screens), but preferential looking is the same as in-person.
    - There is no default attention-getter, you have to make your own.
- Once you save the experiment, it saves as a self-contained folder and you can interact with it like any other PyHab experiment, including modifying it in the builder later.

One big, critical difference in when movies start playing: As mentioned above, in a normal in-lab PyHab study, if a trial is not set to auto-advance, the stimuli will not begin playing immediately when the attention-getter ends. Rather, it waits for the first gaze-on event, and until then, presents the first frame of the movie as a still. If you want to replicate this behavior in an online study, **you need to make a separate trial type for the still-frame that presents it as an image**. I recommend setting this trial type's parameters to "OnOnly" with a very short minimum on-time (say, .1), which will basically mean that as soon as a gaze-on is detected, it advances to the next trial. The next trial should be the movie, and it should be set to auto-advance and have no attention-getter, so it starts seamlessly.

d. Running an online experiment

Running an online experiment is pretty similar to running one in the lab, from the experimenter's perspective. All the keys work just like they do in an in-lab study, and the experimenter window is the same. The main difference is how you get it set up initially, and what's going on with the videoconferencing aspect.

- First, you need to send the parent the link to the presentation. This is the same as the presentation link you put in the stimuli settings.
- This will take them to a webpage that asks them to click a button to continue. Tell them to click the button.
- Ask them to make the browser full-screen. This will hide Zoom from view, and unlike a screen-sharing experiment, there won't be a little mini-portrait with you next to the slides.
- Alternately, ask them to share *their* screen, so you can validate the stimuli were presented correctly. You may also need them to minimize the pop-out window with the webcam views so it doesn't obstruct any of the stimuli (and if it shows their own webcam feed the baby will get very distracted).
    - Is it better to have them share their screen and risk interference from the Zoom pip, or not be able to directly validate the stimulus presentation from the recording? That's your call. Unfortunately I don't have a perfect solution that solves all of the problems, but as long as the parent minimizes the Zoom pip while they're screen-sharing, it should be fairly unobtrusive. You can also ask them to put it in a particular area of the screen where you know nothing important will show up.
- Once the slide-show is open on the participant's computer and PyHab is running the experiment on yours, you can proceed like it's a normal PyHab experiment. All of the data recording is happening on the experimenter's computer, in the experiment's data folder, just like a normal experiment. That includes the timing file, though the timing is based on when the experimenter tells the system to advance, not when it actually

advances for the participant. Expect a small amount of lag from the website in terms of when they see the stimuli, and from Zoom in terms of when you see their response to the stimuli (probably around 200-300ms all told).

- Tips from personal experience:
    - Babies love keyboards. Ask the parent to make sure they can't whack it during the study. Similarly, touchpads on laptops can be a problem if it's in reach. This is a good reason to ask the parent to share screen, but it won't tell you if they pull up the Zoom interface (usually).
    - If the parent has multiple screens, ask them to turn off all of them except the one being used for the study.
    - The most likely thing to lag during an experiment is the Zoom connection, which will slow down your reaction to the baby's behavior. Slides.com is actually pretty light-weight as websites go, and it loads all the media in a presentation at the start, so usually there aren't any major issues with the stimulus presentation itself.

6. 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.

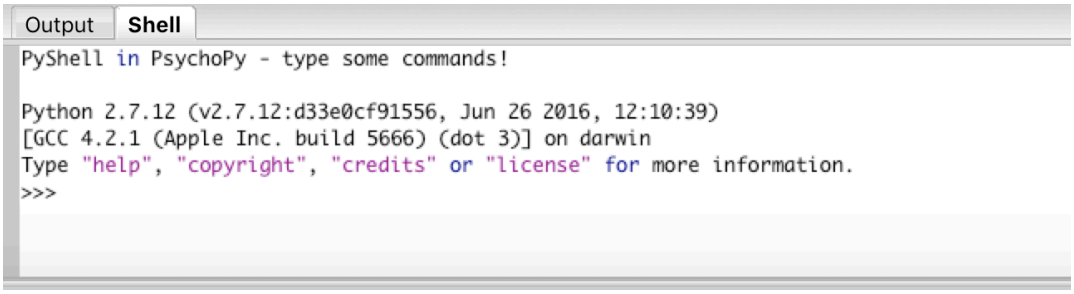
As for everything else, here are some known issues you might encounter:

- **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.
- **Crashes with “no such file or directory” error in PsychoPy3 versions 2020.1.0 or 2020.1.1:** See The PsychoPy 2020.1.0 and 2020.1.1 problem
- **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

Movie playback is surprisingly difficult to program and control. Changes to how movie playback works have introduced some occasionally glitchy behavior for certain experiments on certain operating systems. These failures are very nearly random - some of them only occur on Windows, some only occur on Mac, and some of them only occur on one of those systems with particular movie types! It makes it almost impossible to come up with a true fix, but there are active efforts to improve playback reliability in PsychoPy and hopefully those will address the problem soon.

Until there, here are a couple of things you might encounter and ways you might be able to solve them:

Whole experiment freezes and won't respond to input at the end of a movie playback: This seems to happen for some movie files on some operating systems when there are 10 or more trials, particularly if the trials involve showing the same movie several times. One fix, which may introduce other problems, is the final item in the Universal settings dialog. Basically, by default PyHab tries to conserve memory by loading each movie file only once, and then just 'rewinding' it if it used by multiple trials. The rewinding seems to introduce some problems. This setting changes how PyHab loads movies so that it instead loads a separate copy of the movie for every trial it will be displayed on. This puts a lot of load on your RAM, and may cause the experiment to crash on "Loading Stimuli", but if it loads everything successfully it should fix the freezing problem.

Movies on later trials start in the middle and then snap back to the start: This actually has the same root cause as the freezing behavior described in the last paragraph. The rewind isn't working properly. Same solution, try changing the last option in Universal settings to load multiple copies of each movie file.

Movie displays still-frame properly but then jumps ahead on first gaze-on: This might happen on some systems when there's a delay between the start of the trial and the first gaze-on. The basic problem is that for some reason, some background process starts the movie playing as soon as the first frame is drawn, but doesn't update what's on-screen. Why? Couldn't tell you. The only real solution I've been able to come up with is very much a kludge: Create a trial type that displays images, and give it the first frame of the movie in question. Then set it up so that it is an "on only" trial type, and auto-advances into the movie trial very shortly after the gaze-on key is pressed. This basically just replaces the still-frame that the program *should* provide with a manual still-frame, but because you only start showing the movie when you need it to play, it stops the jumping around.

Other issues: Most other issues you'll encounter have to do more with the movie files themselves than with PyHab. For whatever reason, certain types of encoding cause problems. mp4 files with h.264 encoding tend to be the most reliable, but it's not guaranteed. If your movie is super high-resolution and high-framerate it can make the whole system very slow, especially if

the movie file is also very long. If each individual frame is at 4k resolution and you're displaying 60 of them a second, that's just a lot of data for your system to process! Dedicated movie players make it work by doing a lot of tricks PyHab can't, so try to make your movie resolution only as large as you need and no larger, and if 30fps is good enough, keep it to 30fps.