Pyhab User Manual

v1.5

Winter 2017-2018

Dr. Jonathan F. Kominsky

1. What is PyHab?

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

The single biggest advantage of PyHab over its predecessors is that it can be used to directly control stimulus presentation, and it's free and open-source. Previously, you typically 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. A few other programs (notably Habit2) also allow for this, but are expensive and proprietary.

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 techniques!

2. What do you need to be able to use PyHab?

- You need PsychoPy, first of all (available free from psychopy.org). PyHab is a script that runs in a PsychoPy environment.
- You DON'T need to know python! I've done my best to make it easy for you to build your study in PyHab by modifying a few simple lines of code. Python experience will help you with troubleshooting and allow you to do more ambitious things with PyHab, but the goal is that you should be able to use PyHab with only the instructions in this guide.
- You will need the script itself in a folder. If you are doing stimulus presentation, you will need the attention-getting sound file (upchime1.wav) in that folder, and all the stimulus movie files somewhere you can refer to easily.
- Once you have everything set up, simply open the script in PsychoPy's 'coder' interface (view > go to coder view or command-L on Mac or control-L on Windows) and run it from there.
- To use PyHab for stimulus presentation, you will also need a computer with a two-screen setup. One screen will be your stimulus presentation screen, which will display stimuli to your participant. The other will be the coder's screen, which will display a small graphical interface telling you when a trial begins and ends. The displays cannot be mirrored, you need to have an extended desktop (like you would for a slide show with presenter view).
- If you want to present movie files, you will want your stimuli in .mov, .mp4, or .m4v format for the most reliable performance. PsychoPy's MovieStim works with other formats, but I have not tested all of them with PyHab and I'm not sure how well they work.
- In addition, **you will want to design movies that have about two to four frames of buffer at the end**, and run at either 30 or 60fps. 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 50ms before the end of the movie (roughly 1.5 frames at 30fps, or 3 frames at 60) to guarantee that the movie doesn't unload itself.

- You may need the new and upgraded movieStim script I created for PsychoPy, prior to 1.85.5. Following 1.85.5, the version that comes with PsychoPy should be more current. To install this script, on a Mac go to applications, right-click PsychoPy, select "show package contents". In the folder, go to "Contents > Resources > lib > python2.7 > psychopy > visual", and replace the file named "movie3.py" in that folder with the one downloaded from the PyHab github. On Windows, go to Program Files(x86) > PsychoPy, and in one of the folders in there (I need to get back to a Windows machine to verify which one) you will find lib > python2.7 > psychopy > visual. Again, as of PsychoPy version 1.85.5, this should hopefully not be necessary.
- **On Windows**, you will need ffmpeg installed to be able to use the stimulus presentation features. If you attempt to run a study with stimuli and get an imageio crash, follow these directions: In PsychoPy, open a coder window, then at the bottom select the tab labeled "shell". At the prompt in this tab, type "import imageio" and hit return, then type "imageio.plugins.ffmpeg.download()" and hit return. This should download and install ffmpeg for PsychoPy. If it fails, close PsychoPy and then run PsychoPy as an administrator (right click > run as administrator).
- **On Mac**, you will need to install VLC media player on any computer that you plan to use to present stimuli. VLC serves much the same role as downloading ffmpeg on Windows, it ensures that the right codecs are available on your system for playing movie files. Fortunately, VLC is completely free. Just google it, download and install.

3. How to build your study in PyHab

a. Relevant concepts

Let's define a few important terms:

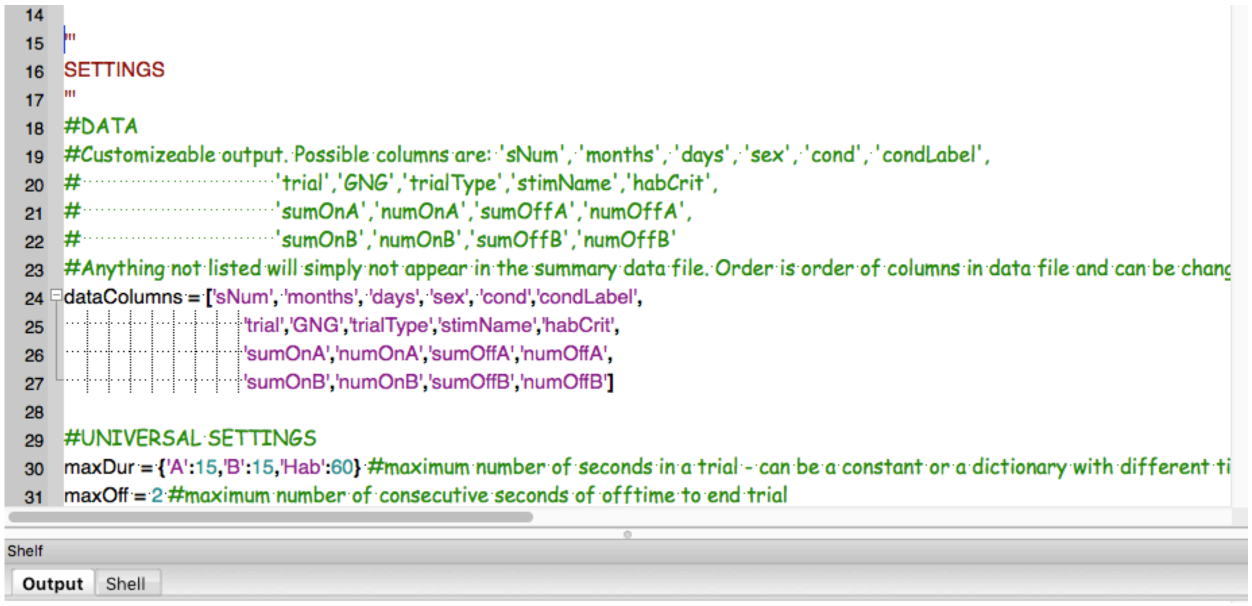
- **Experimenter window**: The window presented to the experimenter running the study, which includes information about the current trial and (optionally) the live status of the coding keys.
- **Stimulus window**: The window on which the stimuli appear, if PyHab is being used for stimulus presentation.
- **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.
    - There is exactly one special trial type that changes the behavior of the program: 'Hab'. The 'Hab' trial type can only have one stimulus file assigned to it, and will create a series of habituation trials until a habituation criterion is met. If you aren't running a habituation study, don't call any of your trial types 'Hab', but literally any other string should be fine.
- **Stimulus file/stimuli**: PyHab expects all stimuli to be in the form of movie files, which you tell PyHab where to find in the settings. You can also run PyHab just as a looking-time coding software with no stimulus presentation, in which case no stimulus files are needed.
- **Attention-getter**: A short sound and 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.

- **Python dictionary or dict**: A data structure in python, defined by being enclosed in curly brackets {}, consisting of key/value pairs. The keys are typically strings contained in single quotes (e.g., ‘A’), followed by a colon, and then the values can be anything. They are required to present stimuli in PyHab, and are described extensively in the section on stimulus presentation settings.
- **Preferential looking version**: PyHab comes in two versions, and the main difference between them is that the one called “PrefLook” is designed to be used with preferential looking designs, i.e. studies in which there are two targets that the infant can look at, and you want to code which one they are looking at. The “normal” PyHab only allows you to code whether they are looking at the screen or not. The preferential looking version does not allow for multiple coders and does not compute reliability (but reliability can be computed after the fact).

b. Parameters and settings

When you open PyHab in PsychoPy’s coder view, it should open a window with a bunch of text. If you don’t know python, a lot of it won’t make sense to you. Don’t worry about it! The only stuff you need to mess with is in the section labeled “SETTINGS”.



```
14
15  '''
16  SETTINGS
17  '''
18  #DATA
19  #Customizeable output. Possible columns are: 'sNum', 'months', 'days', 'sex', 'cond', 'condLabel',
20  #                              'trial','GNG','trialType','stimName','habCrit',
21  #                              'sumOnA','numOnA','sumOffA','numOffA',
22  #                              'sumOnB','numOnB','sumOffB','numOffB'
23  #Anything not listed will simply not appear in the summary data file. Order is order of columns in data file and can be chang
24  dataColumns = ['sNum', 'months', 'days', 'sex', 'cond','condLabel',
25                 'trial','GNG','trialType','stimName','habCrit',
26                 'sumOnA','numOnA','sumOffA','numOffA',
27                 'sumOnB','numOnB','sumOffB','numOffB']
28
29  #UNIVERSAL SETTINGS
30  maxDur = {'A':15,'B':15,'Hab':60} #maximum number of seconds in a trial - can be a constant or a dictionary with different ti
31  maxOff = 2 #maximum number of consecutive seconds of offtime to end trial
```

This is broken into several sub-sections, each with a handful of relevant options.

Data settings

The data settings define how your output looks. The green text is a list of all the possible columns of data to appear in your summary data file. Then, lines 24-27 are the list that will actually appear in your data file, and the order in which they will appear. Each one must be in single quotes, separate by commas. A few columns might not be clear from their names alone:

- 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: Involves condition randomization. More on that later.

Also in this section is the Prefix. This is what will appear at the start of the name of each data file created by PyHab. It's a useful way to keep track of which data file comes from which experiment.

Universal settings

These settings apply to any study you run, whether or not you are presenting stimuli.

- **maxDur**: This is the maximum length of a trial, in seconds. Note that this is NOT maximum gaze-on-time, this is maximum amount of time between the start of trial and end of trial, period. This can take two forms. The simplest form is a number, that will apply to every trial in your experiment. The more complex form is a Python dictionary that allows you to set maximum times separately for each individual (see below).
    - **Advanced:** Python dictionaries are enclosed in curly braces {}, and consist of pairs of "keys" and "values" split by a colon, with a comma between each key/value pair. So if you had trial types 'A', 'B', and 'C', you could create a maxDur dictionary that looked like this: {'A':15, 'B':30, 'C':60}. This would indicate that the maximum duration of trials of type A is 15 seconds, 30 seconds for trial type B, and 60 for trial type C. You need to list EVERY trial type in maxDur if you do it this way, so it's only worth your trouble if different trial types have different maximum durations.
- **playThrough**: Generally speaking, PyHab assumes that your trials are infant gaze-contingent. That is, they will end based on when the infant looks at and looks away from the screen. However, if there are certain trial types that you want to play straight through to their maximum duration regardless of looking behavior, put them in this list. The form of the list is enclosed in square brackets, and each trial type you want to

play straight through enclosed in single quotes and separated by commas (e.g., ['A','B']). If you don't want any non-contingent trials, just set this equal to [ ]. This is useful if you want a movie stimulus to play straight through to the end regardless of looking behavior, you just need to set the maxDur for that trial type to the length of the movie.

- **maxOff**: Maximum continuous gaze-off-time in a gaze-contingent trial, in seconds. For example, if set to 2, trial will end if the participant looks away for two continuous seconds after reaching the minimum on-time.
- **minOn**: Minimum gaze-on-time (not continuous). The maximum off-time only applies after the minimum on-time has been reached, but the maximum total duration applies regardless!
- **blindPres**: This controls how much information the experimenter gets while running the study. It has three possible settings: 0, 1, and 2.
    - 0: 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.
    - 1: 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.
    - 2: 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.
- **autoAdvance**: True or False. If False, the experimenter must manually start each trial. If True, the next trial will start automatically at the end of the preceding trial (playing the attention-getter if there is one, see stimulus presentation settings).
- **randPres**: True or False. If False, then the order of stimuli within each trial type is set and will not change between participants. If True, you must create a condition file (see condition files) but you can make it so that the order of stimuli within each trial type differs for each participant without having to modify the script on every run.

Notably, creating a condition file also allows you to completely blind the experimenter to the condition.

- **condPath**: The folder containing your condition file (if one is needed), enclosed in single quotes. Only relevant if randPres is True.
    - **Note:** On Mac, each subdirectory is followed by a forward slash and the string must end with a forward slash (e.g., '/Users/me/stuff/'). On Windows, each subdirectory is followed by **two** backslashes, and the string must also end with a double backslash (e.g., '\\Documents\\mystuff\\PyHab\\'). The reasons for this difference are irrelevant but it boils down to design decisions made a very long time ago in the architecture of both Windows and Python.
    - **Note 2:** You may find you need to look up how to write out file paths in your operating system. There are some useful tricks. For example, on mac, starting the path with ../ will move it one folder up. If you just write 'folderName/' it will only look inside the same folder the script is located in. If you've never had to use MS-DOS or the Terminal window, this might be a little difficult and unintuitive. You can find some useful info here: https://en.wikipedia.org/wiki/Path_(computing)#Example. The Unix example works for MacOS (which is Unix-based).
- **condFile**: The name of your condition file, which will be a csv file, enclosed in single quotes and ending in .csv (e.g., 'conditions.csv'). Only relevant if randPres is True.
- **condList**: A list of labels for conditions in the condition file, corresponding to the left column of the condition file. Only relevant if randPres is True. See condition files for more information.

Order of presentation

There is only one setting in this section, but it is arguably the single most important setting. The setting is called "trialOrder" and it controls the order in which trials are presented. It is a list, and each entry in the list is an instance of a particular trial type, except for ‘Hab’ which is a habituation block.

- For example, a trialOrder of [‘A’,’A’,’A’,’A’,’B’,’B’] would present four trials of trial type A, and then two trials of trial type B. A trialOrder of [‘A’,’A’,’B’,’B’,’A’,’A’] would present two trials of trial type A, two of trial type B, and then two more of trial type A.
- A trialOrder of [‘A’, ‘A’, ‘Hab’, ‘B’, ‘B’] would present two trials of trial type A and then a habituation block, and when the habituation criterion was met there would be two trials of trial type B. Habituation blocks are special. They consist of a number of trials between two and the maximum set in the habituation settings, and how many habituation trials are present to a given infant is determined by when they meet a habituation criterion. **You only need one instance of ‘Hab’ to insert a full habituation block, and there should only be one instance of ‘Hab’ in your trial order. Multiple habituation blocks are currently not supported.**
- Note that each trial type (except Hab) can have multiple stimulus files associated with it. So, in the examples given above, each instance of trial type A may consist of a different movie file, depending on the stimulus presentation settings.

Habituation design settings

These settings only apply if you have a ‘Hab’ block in your trialOrder. Otherwise they are ignored.

- **maxHabTrials**: The maximum number of habituation trials in a habituation block. If this many habituation trials go by without the habituation criterion being met, the habituation block will end regardless and the next trial type will be presented.
- **setCritWindow**: The calculation for the habituation criterion can be customized. This specific option tells PyHab how many habituation trials to examine when establishing

the habituation criterion. It will sum the total looking time across the first X has trials, where X is whatever you put for this option, and then divide it by setCritDivisor to create the habituation criterion.

- **setCritDivisor**: As described above. The default setting for this is that the habituation criterion is the total looking time of the first three trials, divided by two. For that, setCritWindow would be 3 and this setting would be 2. If you wanted the average of the first three trials instead, you could set this to be equal to setCritWindow. If you wanted just the sum of the looking time over first X trials as your habituation criterion, you can set this to 1.
- **metCritWindow**: This is how many trials PyHab will sum across when determining whether the habituation criterion has been met. That sum is then divided by metCritDivisor, and if the resulting value is less than the habituation criterion, the criterion is considered “met” and the habituation block ends.
- **metCritDivisor**: By default this is set to 1, meaning that the value being compared to the habituation criterion is just a moving window of the most recent metCritWindow trials. However, you can change it if you would like to use a different habituation criterion.

Stimulus presentation settings

If you just want to use PyHab as a replacement for XHab or JHab and present your stimuli some other way, this section is irrelevant. However, if you want to use PyHab to present stimuli so you can directly control the stimulus presentation by coding looking times, this is how you do it. These settings control whether, what, and how stimuli are presented.

- **stimPres**: True or False. If set to False, PyHab functions basically like XHab or JHab, and will not present stimuli. The other settings in this section are therefore ignored. If set to True, PyHab will present stimuli.
- **moviePath**: The folder where the stimulus movie files are located. The same rules apply

here as to condPath in the universal settings, reproduced below.

- **Note:** on Mac, each subdirectory is followed by a forward slash and the string must end with a forward slash (e.g., 'Users/me/stuff/'). On Windows, each subdirectory is followed by **two** backslashes, and the string must also end with a double backslash (e.g., 'Documents\\mystuff\\PyHab\\'). The reasons for this difference are irrelevant but it boils down to design decisions made a very long time ago in the architecture of both Windows and Python.

- **movieNames**: This is a Python dictionary, and a complicated one (sorry). It's easiest to explain with an example. Say you had two trial types, A and B, and each type has two different movies associated with it. For example, in a violation-of-expectation design, trial type A might be two familiarization movies named famMovie1.mov and famMovie2.mov, and trial type B might be two test movies named testMovie1.mov and testMovie2.mov. In that case, movieNames would look like this:

```
{'A':['famMovie1','famMovie2'], 'B':['testMovie1','testMovie2']}
```

Like all Python dictionaries, this is made up of pairs of "keys" and "values". The keys here are 'A' and 'B', which are the trial types, and must always be in single quotes. Everything after the colon is the value that corresponds to that key. In this case, the values are themselves lists of the names of the movie files (without their file extensions). The lists are enclosed in square brackets [], and the names are enclosed in single quotes, separated by commas. The first key/value pair is separated from the second key/value pair by a comma. You can have as many key/value pairs as you have trial types, and each list can be as many movies as you want. There are several important things to note about how this works.

  - **Important Note 1**: PyHab will cycle through a list of trials for a given trial type. So, the first time trial type 'A' comes up in the trialOrder list (see order of presentation), it will play famMovie1. After that trial ends, the next 'A' trial will play famMovie2. After that, the next 'A' trial will play famMovie1, and so on and so forth. The order defaults to whatever order they are listed in on this line, but a condition file can change the order of the list of movies for a given trial type (see condition files).

  - **Important Note 2**: The ‘Hab' trial type is special. Only the first movie in the list will be loaded (PyHab assumes habituation is the same movie presented repeatedly), but you can change which movie is the first item is using a condition file.
  - **Important Note 3**: If there are fewer instances of a trial type in the trial order than there are movies in that trial type’s list, then any movies at the end of the list will be ignored. So, in the above example, if there was only one instance of trial type ‘A’ in the trialOrder list, only the first movie would be seen by participants.
  - **Important Note 4**: To create a between-subjects design without needing separate scripts, you should list every movie that any participant would see in a given trial type and then manipulate the list with a condition file. This is described in depth in the section on condition files.

- **movieExt**: The file extension for your movie files. In the above example, if the movie file itself was named famMovie1.mov, this would be ‘.mov’. You need the leading period and the whole thing to be encased in single quotes. If you have movies with multiple file extensions, make this ‘’ and put the file extensions in the movieNames lists.
- **screenWidth**: The width, in pixels, of your display screen. This should be equal to whatever the width of the resolution on your display screen is, so that it covers the background completely.
- **screenHeight**: The height, in pixels of your display screen. Same rules as width.
- **movieWidth**: The width of the movie file. You will need to get info on your movie file to determine this, but you can also change it so that it displays at a particular resolution. Handy if your movie files are bigger than your display screen, or you want to stretch your movie files to fit your display screen.
- **movieHeight**: As movieWidth, but for the height.
- **screenIndex**: Which screen you want the stimuli to appear on. 0 = the computer’s primary display monitor, 1 = the computer’s secondary monitor. You may need to

experiment to figure out which is which, modern Windows machines and Macs can both be fairly inconsistent about this.

- **ISI**: Inter-Stimulus-Interval, in seconds. Within a given trial, if the trial runs longer than the movie, the movie will loop back to the beginning. If you want to put a pause in between loops, add it here. It will freeze on the last frame of the movie for this duration.
- **freezeFrame**: Basically a minimum inter-trial-interval. If you want the first frame of the movie to appear static on the screen before the movie starts playing at the start of the trial, put that length of that pause here.
- **playAttnGetter**: True or False. PyHab has a built-in attention-getter which consists of a chime and a rotating/looming yellow rectangle. If you would like to use the built-in attention-getter, set this to True. If you have your own attention-getter, it will need to be part of the movie files (or a separate trial type listed in PlayThrough with a short maxDur), and you can set this to False. See "Running a study in PyHab" for more information about the attention-getter.

Condition files

Condition files allow you to change the order of movies in the movieNames dictionary, and ultimately which movies are presented. If you set randPres to "true", you *must* provide a condition file. Condition files have a very specific format, and must be saved as .csv files (you can make these in Excel, simply select "save as" and then select "csv" from the file-type list at the bottom of the save dialog). Condition files consist of exactly two columns.

The left column is the condition labels. These must match, *exactly*, whatever is listed in condList in the universal settings. When you start a PyHab run, an information window will appear, and if randPres is true, at the bottom there will be a drop-down menu with all of the items in condList. You select your condition from this list, and it corresponds to a line in your condition file with that label in the left column. For example, let's say you have three conditions, X, Y, and Z. In the left column of the condition, you would put one on each row (*no* quotes). The labels can be whatever you want, but I recommend using only letters and numbers. Your condList in your universal settings would then look like this: [‘X’,’Y’,’Z’].

The right column of your condition file is a python dictionary (see relevant concepts), very similar to the movieNames dictionary (see stimulus presentation settings), but instead of listing the movie names themselves, it scrambles (or more accurately reconstructs) their order within each trial type. Note that this does not affect the order of trial types themselves. To do that you would need separate scripts, or to change the trialOrder list for each run. Let’s go through a few examples to see how it all works.

Let’s say your movieNames dictionary in your stimulus presentation settings looks like this (pretend this all fit on one line):

```
{‘A’:[‘famMovie1’, ’famMovie2’, ’famMovie3’, ’famMovie4’],
      ’B’:[‘testMovie1’, ’testMovie2’]}
```

By default, the order in which movies will appear for a given trial type is the order in which they are listed in the movie names dictionary. So the first trial of type ‘A’ would present famMovie1, the next trial of type ‘A’ would present famMovie2, the first trial of type ‘B’ would present testMovie1, etc.

Now, say that in condition X we wanted participants to see things in the order in which they are listed here, in condition Y we wanted to reverse the order of the movies in trial type A, and in condition Z we wanted to keep the order for trial type A the same but reverse the order of trial type B.

For the line for condition X, in the right column, you would put the following python dictionary:

```
{‘A’:[1, 2, 3, 4],’B’:[1, 2]}
```

The keys correspond to the trial types, as ever. The lists represent the movies from movieList, each number referring to the order of the list in movieNames. So, in the list for ‘A’ here, 1 corresponds to famMovie1, 2 to famMovie2, etc.[1], and then for ‘B’, 1 corresponds to testMovie1, etc. So, this dictionary would simply be identical to the default.

For condition Y, where we want to reverse the order of the A trials but not the B trials, the dictionary in the right column would look like this:

```
{‘A’:[4, 3, 2, 1],’B’:[1, 2]}
```

As you can see, this would change the order of the movies for trial type A. The first A trial would be famMovie4, the next would be famMovie3, etc.

You should be able to guess what we would then do for condition Z:

```
{‘A’:[1, 2, 3, 4],’B’:[2, 1]}
```

So ultimately, your condition file would look like this (in Excel or other spreadsheet software):

X	`{‘A’:[1, 2, 3, 4],’B’:[1, 2]}`
Y	`{‘A’:[4, 3, 2, 1],’B’:[1, 2]}`
Z	`{‘A’:[1, 2, 3, 4],’B’:[2, 1]}`

And when you ran your study in PyHab, you would select X, Y, or Z from a drop-down menu, and it would correspond to one of these orders of presentation.

Changing *which* movies participants see, in addition to the order

You may have noticed that in the above examples, participants ultimately see all the same movies, just in a different order. What if you wanted some participants to see only familiarization movies 1 and 2, and others to only see 3 and 4? Good news, we can do that with our condition

[1] Programmers will probably wonder why I am not counting from 0 in the condition file, because array indexing always starts from 0 in Python. In short because I felt this would be more intuitive to non-programmers, who I expect to be the majority of PyHab users. The script expects the numbers in the condition file to be one higher than the actual index, and makes the correction.

file. The sneaky thing about the condition file is that it's not actually re-ordering movieNames, it's remaking it from scratch. So, anything not referenced in the right column of your condition file **will not be presented** in that condition.

For example, let's say you want participants in condition X to only see famMovie1 and famMovie2, and those in condition Y to see famMovie3 and famMovie4. **Your movieNames dictionary would be identical to the one provided above**, listing all four movies for trial type 'A' and in that specific order. Your condition file would look like this:

X	`{'A':[1, 2],'B':[1, 2]}`
Y	`{'A':[3, 4],'B':[1, 2]}`

So, in condition X, famMovie3 and famMovie4 effectively do not exist. No matter how many instances of trial type 'A' appear in your trialOrder, it will just cycle back and forth between famMovie1 and famMovie2. Similarly, in condition Y, it will ignore famMovie1 and famMovie2.

You can use condition files to reconstruct your movieNames list however you want, but movieNames in the script itself needs to contain each and every *possible* movie for a given trial type. You can't swap movies between trial types and you can't change the order of trial types (i.e. trialOrder) from a condition file.

Notably, this trick allows you to have multiple different habituation conditions with a single PyHab script if you are running a habituation study. 'Hab' trials only look at the first movieName in their list, so you can list all of your different habituation movies in movieNames, and then select which one you want to present in your condition file (e.g., {'Hab':[3]…} would the third movie listed next to 'Hab' in movieNames your habituation stimulus).

If this is a little too much, I highly recommend experimenting with the demo materials. I included a condition file in them, and if you set randPres to true, PyHab will use that condition file. You can then change things in the condition file and see how it affects which movies are presented.

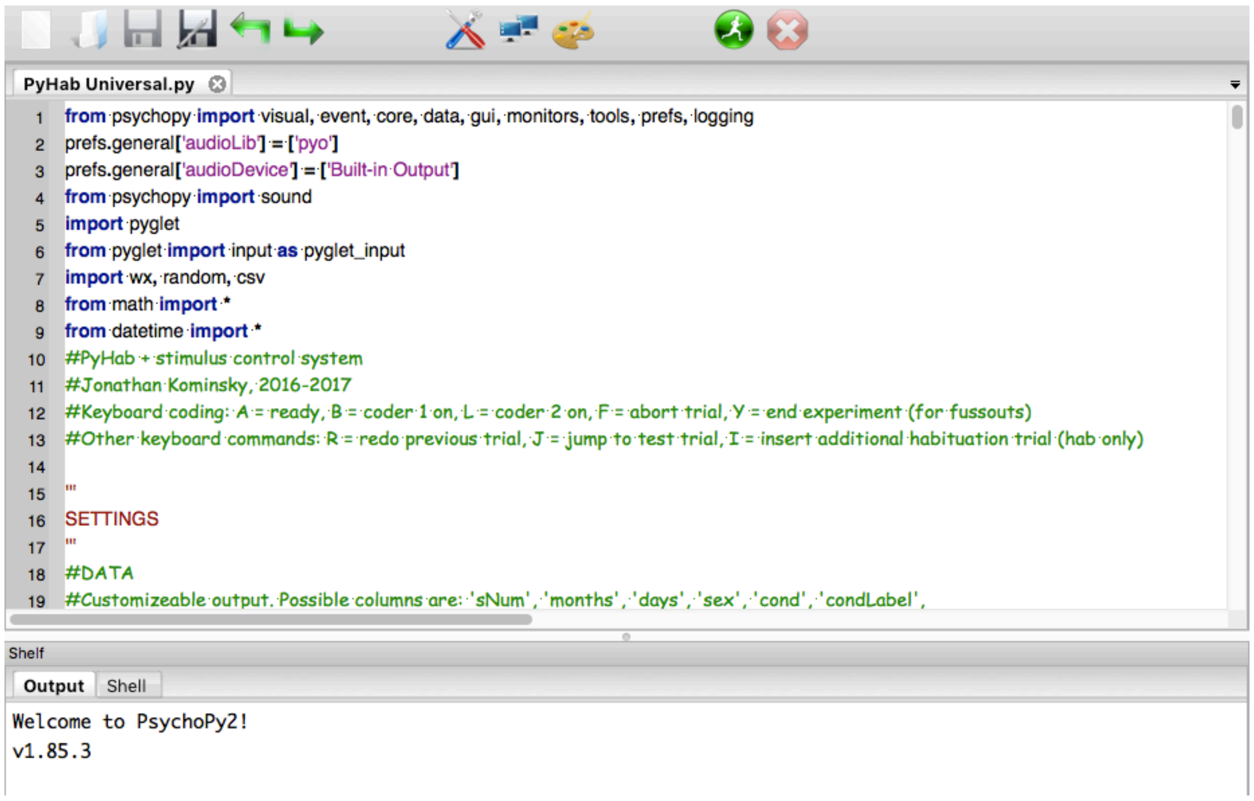
One troubleshooting note: If you put in a number that is greater than the number of items in the movieNames list for that condition (e.g., if we put a 3 for B in the above), the program will crash when you try to run it. If you start using a condition file and get an error that includes “Index out of range”, that’s probably why.

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

When you open PyHab, it will be in a PsychoPy window, and look like a bunch of code. To start the program running, click the big green button in the top bar with the silhouette of a running person. If, for whatever reason, something crashes, use the big red button next to it to kill the program.

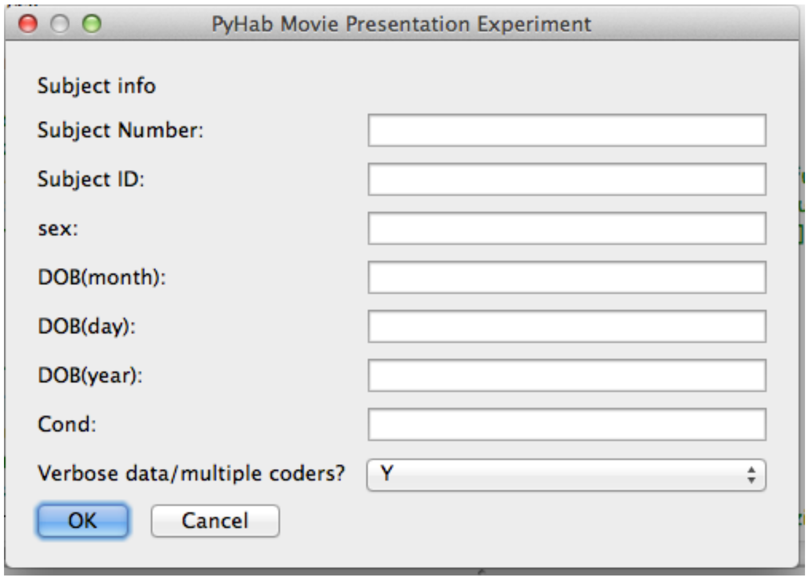


```
from psychopy import visual, event, core, data, gui, monitors, tools, prefs, logging
prefs.general['audioLib'] = ['pyo']
prefs.general['audioDevice'] = ['Built-in Output']
from psychopy import sound
import pyglet
from pyglet import input as pyglet_input
import wx, random, csv
from math import *
from datetime import *
#PyHab + stimulus control system
#Jonathan Kominsky, 2016-2017
#Keyboard coding: A = ready, B = coder 1 on, L = coder 2 on, F = abort trial, Y = end experiment (for fussouts)
#Other keyboard commands: R = redo previous trial, J = jump to test trial, I = insert additional habituation trial (hab only)

'''
SETTINGS
'''
#DATA
#Customizeable output. Possible columns are: 'sNum', 'months', 'days', 'sex', 'cond', 'condLabel',
```

```
Welcome to PsychoPy2!
v1.85.3
```

When you hit the button, 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:



Subject info

Subject Number:	
Subject ID:	
sex:	
DOB(month):	
DOB(day):	
DOB(year):	
Cond:	
Verbose data/multiple coders?	Y

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 overwrite the existing file! You can't leave these fields blank 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, which is used to compute their age at the time of the study run. You need to input the month, day, and year separately, each one in **two-digit** form. 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 in the DOB(year) field. DO NOT put the four-digit year, it will give you a very strange age calculation.[2]
- 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.
- Verbose data/multiple coders is a drop-down list of Y or N that defaults to Y. If it's set to Y, it will produce a "verbose" data file that includes every single look and look away. If set to Y it also allows a secondary coder, and if one is detected it will compute inter-rater reliability. If it is set to N you will **only** get the summary data file.

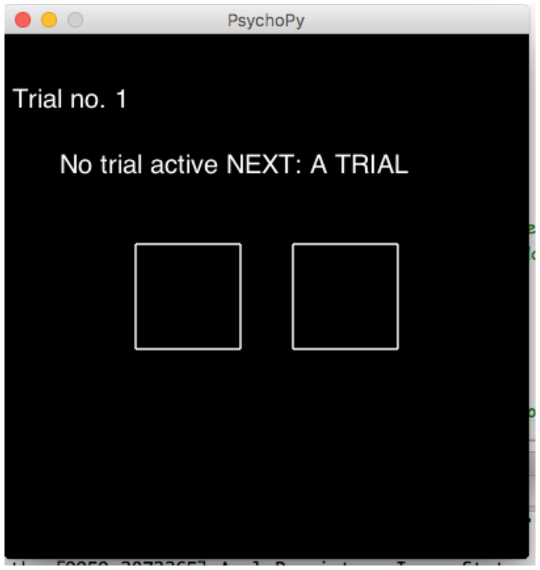
[2] If you are running this program in the year 2100 or later (over 80 years after its initial creation), the age-at-run computation code may need to be modified for people born in the 2090s. Also holy sh*t.

- If you are not using PyHab for stimulus presentation, there will be three further fields not shown on this screenshot, 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.

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

b. Running a PyHab study

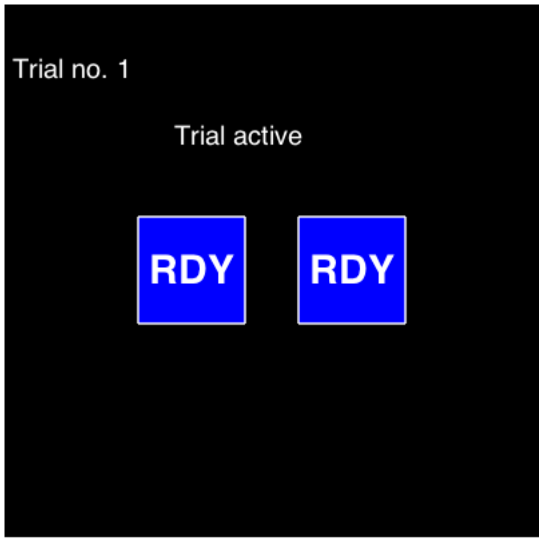
Once you have hit the "OK" button, if PyHab is being used to present stimuli, the stimulus window will appear on the stimulus presentation screen. Whether or not it is being used to present stimuli, a window will pop up on the experimenter screen that looks something like this:



Trial no. 1

No trial active NEXT: A TRIAL

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 universal 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, PyHab will save the data and (attempt to) close. For various reasons having to do with PsychoPy, Python, and certain operating systems, it will sometimes get stuck while closing and you will have to hit the red “stop” button to close it down completely. The data will still be saved, however.

Key commands

Key	Function
A	Play attention-getter (if PyHab's attention-getter is enabled) and ready trial, or start it if auto-advance is enabled.
B	Primary coder gaze-on (or gaze-on Left for pref. looking). Hold down as long as the infant is looking at the screen, release when they are looking away. If auto-advance is not enabled, the trial will start when this key is pressed for the first time.
L	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 the last trial, marking the previously recorded trial as no-good.
J	If you are using a habituation-dishabituation design, you can press this between trials during the habituation block 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-dishabituation design and are right before the test trial (because the criterion has been met or because of the number of habituation trials presented), this key will insert another habituation trial.
P	Print trials so far to the output window in PsychoPy. Only works when not presenting stimuli, and only works in-between trials. Will not work if auto-advancing.
M	Preferential looking version only. Gaze-on Right. Preferential looking only supports one coder, so this key can also mark the beginning of the first gaze-on to start a trial. In every way identical to the B key, except indicating that the participant is looking at the other side of the screen.

c. Data and computing reliability

Data

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

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

criterion.

- 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.

- Verbose data file A: The primary coder's "verbose" data file is saved if verbose is set to Y when the study is run. 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 a verbose data file B is created, then PyHab will also attempt to compute reliability statistics comparing the two coders, which will be output to this file. There are four reliability statistics:
    - Weighted percentage agreement: Percentage of frames on which the two coders agreed, weighted by the length of the trial.

- Cohen's Kappa: A reliability statistic between 0 and 1, preferred by some journals and researchers.
- Average Observer Agreement: The raw percentage of frames on which the two coders agreed.
- Pearsons R: Standard correlation coefficient.

Standalone reliability

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

d. Preferential looking studies

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

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

Everything else is the same between the two versions.

5. Troubleshooting

PyHab has its share of quirks. Because you do have to directly modify parts of the script, it's possible to enter information incorrectly and get errors when you try to run the program. Here, I will catalog some of the errors you might run into, and their most likely solutions. There are also some things that are bugs in Python or PsychoPy, which currently don't have a solution and I'm not in a position to fix.

- **Program crashes right after pressing the green button, says "SyntaxError: Invalid syntax" in the output window in PsychoPy**: This is fairly straightforward. It just means that you forgot to close a bracket or a quote or something while modifying the code, or introduced a stray character. Right above the SyntaxError line will be some blue text, which you can click on to bring you to the affected line. Sometimes it will be a stray key-press that made its way into the code, but often it will not be that line that is the issue but the one before it. For example, if it says line 28 (the prefix setting) is the issue, that probably means you forgot to close something in the data list. Look for a missing close-bracket or close-quotation-mark. This error may also arise if you put a single backslash when specifying condPath or moviePath on Windows. Make sure you put a double backslash!
- **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 Python and MacOS, and it's relatively harmless. Just ignore the error dialog that pops up and try running it again. It sometimes takes two or three tries but it does eventually work.
- **Immediately after entering participant info and hitting OK, the program crashes**: If it doesn't open a window at all, and just crashes as soon as you hit OK, it's almost always due to the DOB or DOT being entered correctly. Leaving other fields blank won't make the program crash outright, but the first thing it does, before evening opening the experimenter window and stimulus window, is try to compute the

participant's age. If that calculation breaks down for whatever reason, it will crash then and there. Make sure you are entering TWO-DIGIT values for each line of the DOB and DOT, and make sure you're not getting month/day/year mixed up and putting in an impossible value for one or the other. No letters, no symbols, don't leave any DOB or DOT line blank.

- **The experimenter and stimulus windows flash up onto the screen briefly and then vanish**: This can be a few different things, but usually only happens when using PyHab for stimulus presentation. It usually comes down to moviePath, movieNames, or movieExt. Your output pane will probably have a bunch of warnings, but if you scroll up it will show you the error.
    - If the error says "IOError: Movie file [name] was not found", then you've entered the path to one of your movies incorrectly. That might mean you've listed the wrong folder in moviePath, the wrong filename in movieName, or the wrong file extension somewhere. There's no easy fix to this, just look carefully for typos or misformatted information. This might help: https://en.wikipedia.org/wiki/Path_(computing)#Example
    - If you are using a condition file and it gives you "KeyError", that probably means that there's a mismatch between condList and the left column of your condition file. Whatever appears in condList needs to be *exactly* identical to whatever is in the left column of your condition file, but inside single quotes.
    - If you get a MemoryError, things get complicated. PsychoPy's code for presenting movie files is rather inefficient, especially if those movie files have audio. I've gone to great lengths to make PyHab as memory-efficient as I possibly could, but it still has to load all of the movies for a given run up front, and if the cumulative memory load is too much, the program will crash. Make sure there are no background programs running on the computer, try to use a computer with a lot of RAM, and make sure your movie files are efficiently encoded using MPEG-4. Remember that PyHab will loop the movie for you, so if your movie is just the exact same event repeating multiple times, you only need the movie file to contain

one repetition.

- **PyHab starts and runs some trials but reliably crashes on a specific trial type**: This is nearly always due to the trial type not being referenced either in movieNames, the condition file, or maxDur. Basically if your trial order includes trial type 'Z', if 'Z' is missing from your movieNames dictionary, or does not appear in the dictionary in the right column of your condition file, or you made maxDur a dictionary rather than an integer and 'Z' does not appear in it, when the program gets to that trial type it will look for information that does not exist. Alternately, there may be something wrong with the movie file for that particular trial type. Try moving the movie to a different trial type and seeing if you get the same crash.
- **PyHab runs the experiment successfully, but hangs at the very end, either when closing the windows, or the windows close but the red stop button remains highlighted and the green button is grayed out**: If it fails to close the windows, and you had two coders, that's probably an issue with the reliability calculation. I have very carefully and systematically gotten rid of every infinite loop I was able to create, but there's a chance one slipped by. If it closes the windows, this is another PsychoPy issue that can't really be fixed. Again, more common on MacOS than Windows. In either case, just hit the red stop button. Your data will be saved, though it won't save the reliability file if it gets stuck in a loop on reliability.
- **At any point, it crashes and displays a "permission error"**: This is a Windows problem. Make sure you run PsychoPy as an administrator (right click on the icon and select "run as administrator" or change the settings on it 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 make it.