Open Monkey Mind

# List of fixed common independent variables

* ~~Location:~~
  1. ~~Country~~
  2. ~~City~~
  3. ~~LabName~~
  4. ~~Id of scientist~~
* Experimental trial
  1. ~~Id of experiment~~
  2. ~~Id of testing unit (number, location, etc)~~
  3. ~~Date and time of the trial~~
  4. Duration (from first trial screen to end of trial)
  5. ~~Reward delivery (yes vs. no)~~
  6. Correction procedure (yes or no; see correction procedure below)
  7. Number of times each given block has been repeated (for instance before reaching the training criterion)
  8. Duration of the time out after an error trial.
  9. Duration of the inter-trial interval
* Data

1. ~~Score (0 or 1 or NA)~~
2. ~~Response time (in ms or NA)~~

* Participant
  1. ~~Id of participant (name, number)~~
  2. ~~Age (in days)~~
  3. ~~Sex~~
  4. ~~Grouping variables 1, 2, 3, 4 (three variables to group the individuals: for instance matriline, social group, enclosure, family, etc).~~

# List of useful functions

## Setup the experiment

**Block randomisation and criterion:** As in opensesame it will be useful to have a simple way of setting the randomisation of blocks (yes vs. no) and to determine under what condition they progress to the next condition (e.g. percentage success on last n trials, or more than x% correct in each of the last n blocks, etc).

**Reward delivery:** It requires at least three parameters: (i) a probability to deliver a reward, (ii) a parameter for the quantity of reward and (iii) a system to detect if there is a reward delivery system attached (in case you are testing the program on a machine without a delivery system attached, in which case it could produce a bip sound for instance).

**Correction procedure:** Sometimes when a trial is failed it is re-presented again, ideally not immediately but within the next three trials. This number of re-presentations of the trial is often fixed to a given number (often 3). So, repetitive failures will produce a maximum of 3 correction trials.

**Aborted trials:** Trials that have not been completed, typically no response with a certain delay, should be saved and coded as an “aborted trial”, and then represented when the subject reinitiate the trials.

**~~Test mode:~~** ~~At present when we want to test programs we have a test-user named "0" and all the trials with 0 are test trials, these are initialised on the 'client side' with a special command 'ctrl+shift' that bypass the chip-reading system. Test user can also be assigned several microchips so that we can test the entire system (Id, response, etc) using testing microchips. I.e. a dummy mode with a fake participant and data logging that is clearly distinguishable from real data.~~

**~~Escape keystroke:~~** ~~It’s also useful to be able to abort a trial while it is running using a special keystroke (e.g. escape). Trials aborted like this are not recorded.~~

## Experimental modes

**~~Pre-loading experiments:~~** ~~For experiments with large files it can be useful to pre-load the experiments on the client side by downloading all the files (sound, pictures, etc…) before starting.~~

**~~Mode local vs. network:~~** ~~One important feature of the software is that the test program can run locally on a single computer. It is important to keep that option if we want, for instance, to install a single computer somewhere in the field or in a zoo with no internet connection. It should be possible to start the server locally, and test it like that. To make this feasible, also for non-technical users, it's important that starting the server is just a matter of clicking a button.~~

**~~Auto-response mode~~**~~. This corresponds roughly to the autoresponse mode of eprime. The program automatically mimics a behavioral response on each trial, in such a way that we can automatically run the full sequence of trials to make sure that everything is correct (this might already be implemented in opensesame?).~~

**~~Network failure task:~~** ~~We also have a task that comes up if the network fails, it’s a simple red square that appears at the centre of the screen and that the monkeys have to touch to get a reward. In our current system, the program checks the connexion to the network at the beginning of each trial, so that the program can abort the “failure task” and resume testing once the computed is again connected.~~

**~~Training tasks:~~** ~~We also have a set of training tasks that each monkey need to go through before they can do experiments. Their difficulty increases until they reach the final stage of training, they are then available for other experiments. If a new ID appears, the monkey is sent automatically to these tasks without altering the course of other experiments. Reserved high priorities for these tasks? Or special status?~~

**~~Background tasks:~~** ~~In the current setup we have a set of tasks that the monkeys default to when they have nothing else to do. It would be great if we could define some tasks with a special status and they would be randomly picked if nothing else is available. Would setting a very low priority achieve that? Or do we need a special status?~~

## Follow experimental progression

**Tools to track experimental progression:** While the experiment is running we need some information on the progression, such as the experimental stage the monkeys are at, the number of trials they have done, their average success rate per block for instance. What we have done in our program is that this information is different from the data, it is saved in a special place, is easily accessible and automatically updated every time a trial is done. This avoids having to download the data every time we need check. Do you think something like that be possible?

**Tools to change the experiment while running:** At present we sometimes have to change the course of an experiment online (while it is running), for instance during training if some monkeys are stuck at a level we might need to bring them back to an earlier level to boost motivation. We also have sometimes bugs when one experiment is abruptly shut down, say if the server crashes the experiment is reset to start and we need to bring back the monkeys to their earlier level when rebooting. The two main routes to do that will be to 1) download the jobs data, edit it in a spreadsheet editor, and re-upload it [easy but not so convenient]; and 2) to modify the job data programatically through the Python API [requires some coding but more convenient].

**Tools to track which subject has initiated the trial in case of errors**. In case of trial bugs, it is always useful to know the name of the last subject who has initiated the trial that produced the bug. Our current program records a file in each client computer, indicating the name of the last subject which has initiated the trial.