Creating Experiments Using OpenSesame

Edwin Dalmaijer

After doing this tutorial, you should...
...have achieved a basic understanding of OpenSesame's user interface
...know how to present questionnaires using OpenSesame
...be able to create a basic response time task (and other types of experiments)
...be able to do a little inline scripting
...know where to look for help on creating experiments using OpenSesame

Introduction

OpenSesame (Mathot, Schreij, & Theeuwes, 2012) is a relatively new, open-source experiment builder for the social sciences. It is cross-platform software, usable on Windows (tested are 2000, XP, Vista, 7 and 8), Mac (not entirely stable on all releases) and on Ubuntu/Debian Linux. OpenSesame is even included in NeuroDebian (Halchenko & Hanke, 2012), and a version that allows you to run experiments on Android devices is available via Google Play.

OpenSesame is Python (Van Rossem & Drake, 2011) based, which brings about two major advantages: support for Python inline scripting and the possibility to create custom add-ons (plugins) scripted in Python (more on both later on). The source code is freely available, adjustable and redistributable under the GNU General Public License. In terms of lay-out and general idea, OpenSesame is comparable to Presentation (Neurobehavioral Systems), Experiment Builder (SR Research), Inquisit (Millisecond Software) and E-Prime (Psychology Software Tools). If you have experience with any one of these, you will most likely be able to use OpenSesame without a lot of getting used to.

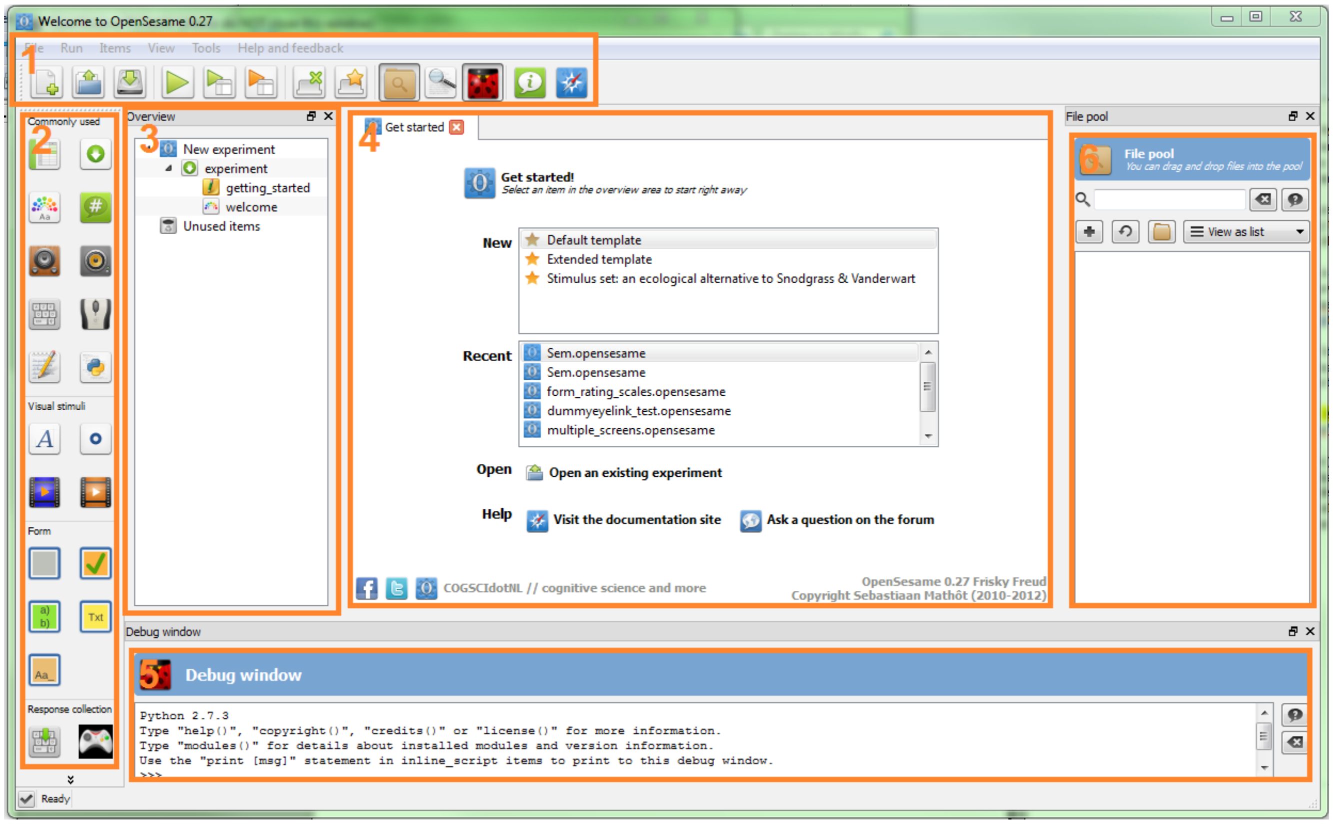
In this tutorial, you will learn how to use OpenSesame's graphical user interface (GUI) and inline scripting functionality by creating three different experiments. You will be provided with a how-to on creating your own plugin and a summary of all the locations you can turn to for help on creating experiments in OpenSesame.

Getting OpenSesame

In theory, you could use any version of OpenSesame you would like (see here for a selection). If you want a version with all the relevant plugins already included, you could use OpenSesame Portable (version 0.27.1), which is available here and can be directly downloaded by clicking here.

Introducing the OpenSesame GUI

Start OpenSesame. You should see something similar to figure 1:


Welcome to OpenSesame 0.27
File Run Items View Tools Help and feedback
1
2
3
4
6
Commonly used
Visual stimuli
Form
Response collection
Overview
New experiment
experiment
getting_started
welcome
Unused items
Get started
Get started!
Select an item in the overview area to start right away
New
Default template
Extended template
Stimulus set: an ecological alternative to Snodgrass & Vanderwart
Recent
Sem.opensesame
Sem.opensesame
form_rating_scales.opensesame
dummyeyelink_test.opensesame
multiple_screens.opensesame
Open
Open an existing experiment
Help
Visit the documentation site
Ask a question on the forum
COGSCIdotNL // cognitive science and more
OpenSesame 0.27 Frisky Freud
Copyright Sebastiaan Mathôt (2010-2012)
File pool
File pool
You can drag and drop files into the pool
View as list
Debug window
Debug window
5
Ready

```
Python 2.7.3
Type "help()", "copyright()", "credits()" or "license()" for more information.
Type "modules()" for details about installed modules and version information.
Use the "print [msg]" statement in inline_script items to print to this debug window.
>>>
```

Figure 1 – *General lay-out of OpenSesame's GUI (see text for numbers and highlighting explanation)*

In this figure, the following crucial parts are highlighted:

1. Menu	Here you will find all the controls for OpenSesame's settings and for saving and running experiments. Pay close attention to the button with the big green arrow (press it to run an experiment) and the button with the arrow pointing down to a disk (for saving the current experiment).
2. Items	Here you see a list of all of the items currently available in your installation of OpenSesame. For more information on the items, see Appendix A: List of Items and their Functions.
3. Overview	This is the 'timeline' of your experiment (which should look pretty familiar to E-Prime users in particular). It lists the experiment's items in a way that shows the experiment's process: the items are run in order from top to bottom, where items with the same indentation are run within the same sequence.

4. Properties	This is one of the most important parts of the GUI. It shows the properties of the item that is currently being selected (or, on startup: it shows the startup options). Here you can adjust the properties of an item.
5. Debug Window	The debug window is a custom, simple Python console in which (debug) messages can be shown and you can type Python code in there to do what you think fit. OpenSesame supports an IPython window as well (for more information, see here).
6. File Pool	Each experiment (if it is saved as an .opensesame.tar.gz file) carries it's own file pool. You can add images, sounds or movies to be used by the experiment, and you can even save logfiles in there.

At startup, you can select a template (with a really basic option, an extended basic option with pre-created loops, or with an entire stimulus set), a recently opened experiment (simply select one from the list), or an existing one (by browsing). To create a new experiment, select the default template. That last bit was not just a general remark, but more of a forceful suggestion: Please select the default template, as you are going to begin creating your first OpenSesame experiment now.

Experiment 1: Creating a Questionnaire

Now you have selected the default template, look at the properties window. You should see the general properties of your experiment (if not, click on the 'New experiment' item on the top of your overview). First, you can rename the experiment to something like 'questionnaire demo' by clicking on the text in the top of the properties window (where it says 'New experiment'), typing in your new name and then pressing Enter. You can also change the description by clicking on the text directly under the experiment's title (where it says 'Default description'), typing in your new description and pressing Enter. The same principle applies to all items.

Apart from the experiment's name, a number of relevant properties are shown. Most of these (e.g. the resolution, default foreground and background colours, and the font) are self-explanatory. The back-end feature probably looks less familiar. The back-end is the Python package OpenSesame uses to create an experiment. If you leave it on the default option, expyriment, you should be fine (but see box 1 for further information).

Set the resolution to values of your liking, the foreground colour to 'black' and the background colour to 'white' (or pick some other fancy colour using the box with the differently coloured squares, next to the text input). Also, it might be a good idea to reduce your text size a bit (since the default 18 could be a tad big for questionnaire items).

xpyriment	Based on expyriment, a python library created for creating cognitive and neuroscientific experiments. Expyriment has excellent timing capabilities for visual stimuli (see here for benchmark tests), thanks to OpenGL.
legacy	Based on PyGame, a python package for creating games. This may sound weird, but do keep in mind that game designers look for the same software properties as psychologist do: proper processing of input and output, with minimal delay. This is the only back-end that does not use OpenGL routines, therefore it is suitable for less advanced computers (you could even run experiments build in this back-end on a Raspberry Pi, a $25 computer). Be wary that the legacy back-end has a less predictable timing; see Mathot, Schreij and Theeuwes (2012) for a comparison of the legacy and psycho back-ends.
psycho	Based on PsychoPy (Pierce, 2007 and Pierce, 2009), a rather renowned package among Python-using cognitive neuropsychologists. It is very comparable to the Psychophysics Toolbox for Matlab, a.k.a. 'the PsychToolbox' (Brainard, 1997). PsychoPy uses OpenGL for visual presentation, like expyriment, and has millisecond-accurate timing properties for visual stimuli. The main difference with the xpyriment back-end is in the Python functions underlying OpenSesame, that you do not have to deal with if you stick to using the GUI. The advantage of having both packages aboard, is that you can use both expyriment and PsychoPy functions in inline_scripts.
droid	Based on the PyGame Subset for Android, which is a version of PyGame that works on the Android platform. Using this back-end, you can create experiments that can be run on Android phones and tablets on which the OpenSesame Runtime (available via Google Play) is installed. Please note that the OpenSesame app does not support experiment building; it was designed to run experiments only. You can simply build an experiment on your computer, and then save it to your phone or tablet.

Box 1 – *OpenSesame's back-ends (as of version 0.27)*

sequence item

A sequence item is much like a sub-timeline in the experiment, where a number of other items can be stored in to run, as one would suspect from the name, in sequence. If you look at your overview, you can see one already: it is called 'experiment' and you can recognize it by the icon with the green circle and the white downwards arrow. Please click on it and look at the properties window.

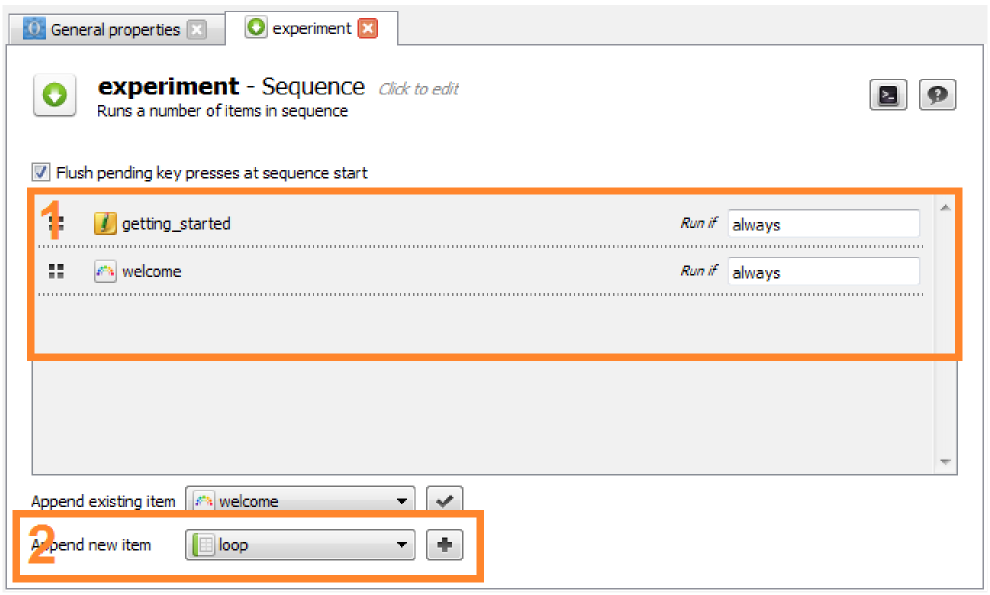


Figure 2 – *Properties window of a sequence item (see text for numbers and highlighting explanation)*

Figure 2 shows what you should be seeing right now. At 1 the items within this sequence are shown. At 2 there is a button for adding new or existing items. The items currently listed are a notepad and a sketchpad. The former does nothing for the experiment flow, but is used for commenting (use it as a 'note-to-self' or to other users of the experiment). The latter is one you will be seeing a lot, as it is the main item for presenting static visual stimuli (more on this later). For now, we are going to delete only the sketchpad (called 'welcome') by clicking on it with the right mouse button and selecting 'Delete'.

TIP! Beware! If you delete an item from the experiment's overview, it still exists. It is now stored in the 'Unused items'. You can find these by clicking on the little arrow left of the unused items bin, at the bottom of the overview. If you wish to completely delete an item, click on the unused items bin in the overview and click on the button labeled 'Permanently delete unused items'. This could be crucial, since terribly messed up unused items may still cause your experiment to crash.

Before a participant starts the actual experiment, you want to make him or her agree to participate. OpenSesame has a default informed consent form[1], which can be added to the experiment by clicking behind 'Append new item' (see figure 2, number 2) in the properties window of your sequence, selecting 'form_consent' and then pressing the '+' button. If you do so, the newly added consent form should appear below the other item.

To see what it does, let's do a testrun. Press the button with the big green arrow, the run-button, in the menu. Enter a subject number (0 seems appropriate for testruns), select a location to save the logfile, and then the experiment starts. There should be a loading screen, and a form to read and accept. After the completion of this form, the experiment should quit. Afterwards, OpenSesame asks if you want to copy the logfile to the file pool. This might be useful for actual data, but not for testruns (logfiles are stored on the previously selected location anyway, this is just an option to keep copies of your data files along with your experiment).

1 Please note that some journals require **written** informed consent!

Now that you have informed consent, you are going to inform participants about the experiment. Do this by appending a form_text_display item (go to the sequence, click right of 'Append new item', select the form_text_display and press the '+' button). Click on the new item to adjust it. You can change the name (to something like 'intro'), the form title (shown to a participant) and the message that you want to present to the participant.

Next up, you can pretend to care about your participant's feelings, by asking him how he feels. Do so by appending a form_multiple_choice. Click on the new item to adjust it. Again, you can change the name of your item (shown in the overview), the form's title (shown to the participant), the question you want to pose to your participant ("How have you been?"), the answer categories ("Good", "Kind of ok", "Terrible") and whether you want to allow multiple responses (if not, untick the box labeled "Allow multiple options to be selected"). Another interesting feature is the possibility to have your experiment advance immediately once a participant has picked an option, instead of after pressing a confirmation button. This could speed up your experiment, but prevents participants from correcting a mistake. Finally, you can choose the name of the variable to store the response in. Per default, this is called 'response', which does seem like an appropriate name.

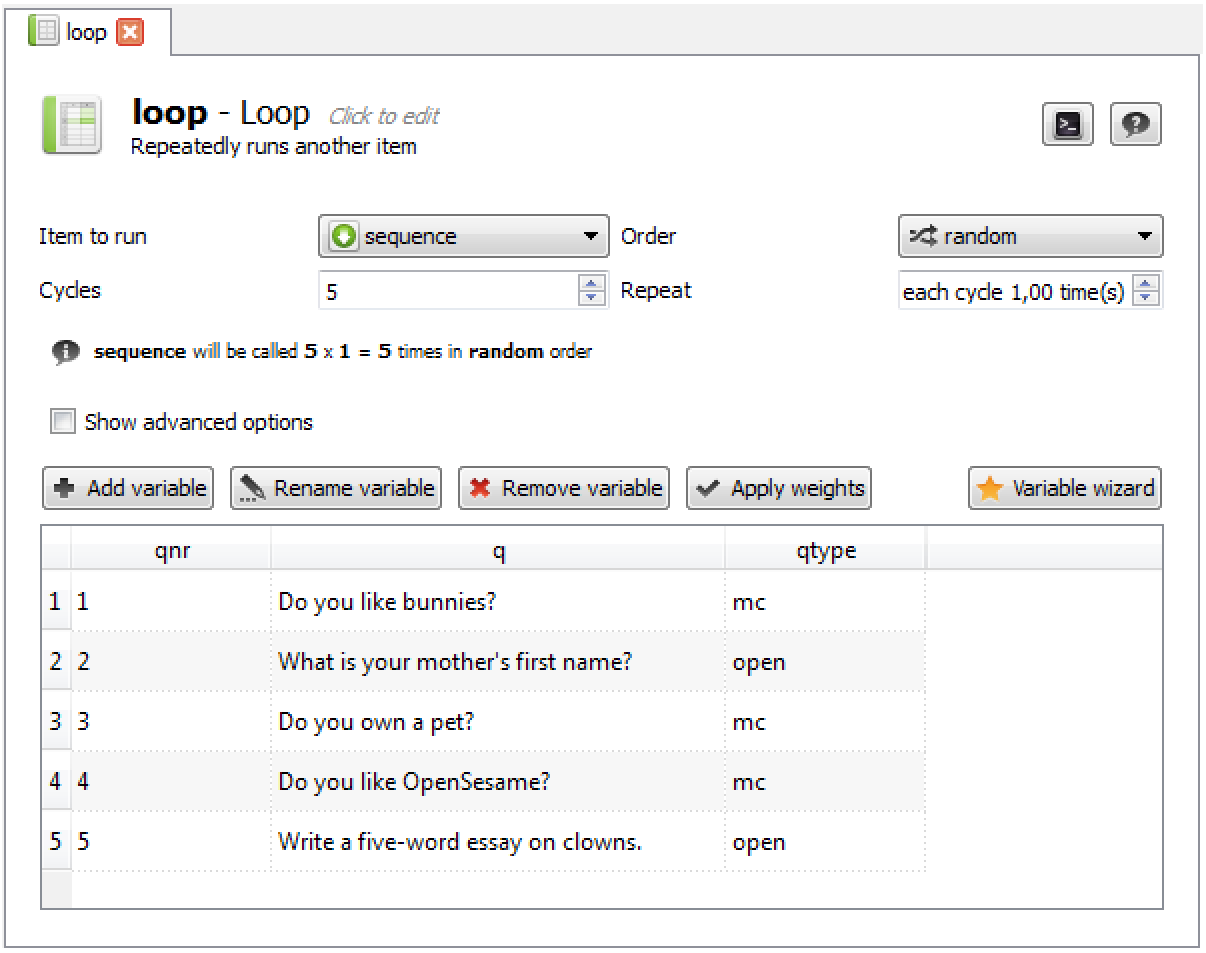
Run your experiment again to see what happens.

creating trial blocks
As you can imagine, adding another 300 form_multiple_choice items to create a questionnaire is a bit of a nuisance. Luckily, OpenSesame has a nice and simple way of creating trialblocks. Click on the experiment sequence and append a loop item. OpenSesame will now tell you that a loop item needs another item to run and that this other item is usually a sequence. Who are we to doubt OpenSesame? Select a sequence under 'Create new item to use' and click on 'Create'. Open the loop by clicking it. If you have used E-Prime in the past, this should look familiar.

In figure three you can see an example of a loop item. The loop's Cycles property determines the amount of unique trials. The Repeat property determines how often these unique trials are repeated and the Order property determines whether the cycles are run in a sequential or in a random order. The most important thing to note, are the variables, of which the names are displayed in the columns' headers. Each row is a different cycle, in which the variables' values for that cycle are defined (e.g. value 'open' for the variable 'qtype' in cycle 2). These variables can be referred to in items by using square brackets, like so: [qtype]. We will be using that information shortly.

In most experiments, a single cycle would be a single trial. So in each row, the properties for a single trial are defined.

adding variables
To add a variable, click on the 'Add variable' button. Please copy everything you see in figure 3 to your own loop. You will have to do this manually this time, although OpenSesame does support direct copy-pasting from spreadsheet editors (e.g. Microsoft Excel and OpenOffice Calc).



Item to run: sequence
Order: random
Cycles: 5
Repeat: each cycle 1,00 time(s)

sequence will be called **5** x **1** = **5** times in **random** order

	qnr	q	qtype
1	1	Do you like bunnies?	mc
2	2	What is your mother's first name?	open
3	3	Do you own a pet?	mc
4	4	Do you like OpenSesame?	mc
5	5	Write a five-word essay on clowns.	open

Figure 3 – *Properties window of a loop item*

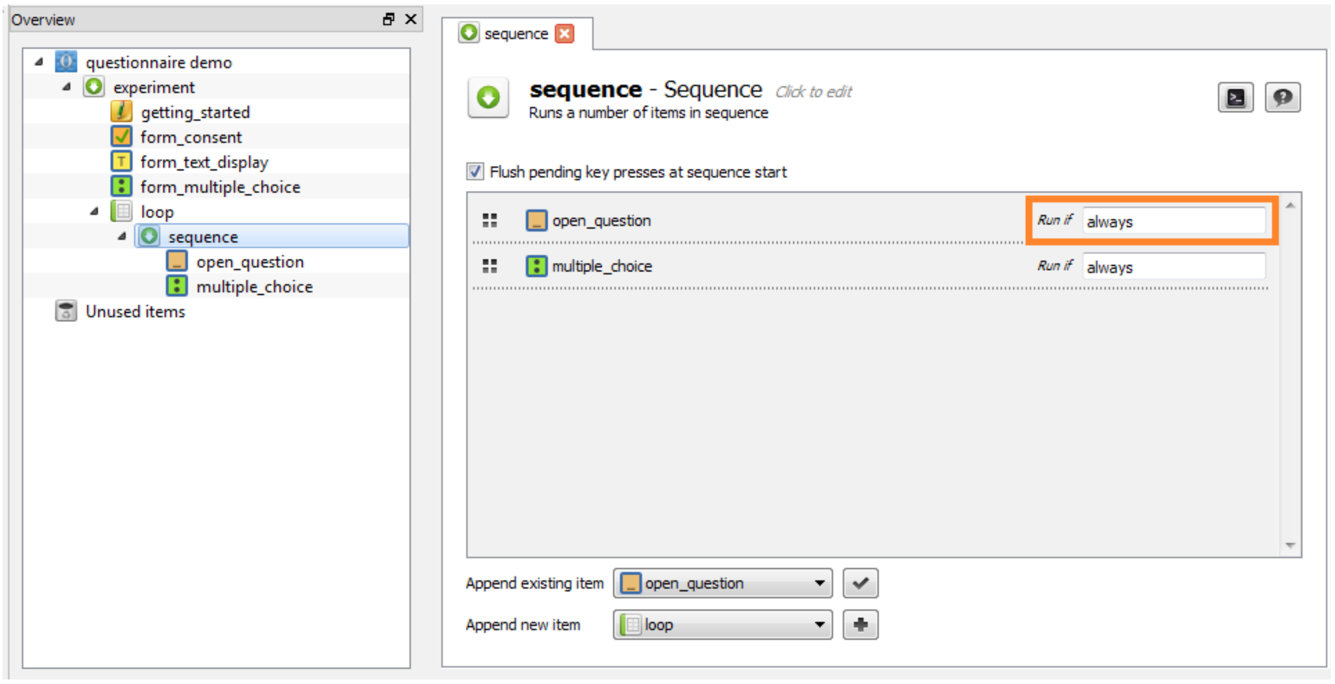
referring to variables
In the overview, click on the sequence item that is shown directly below the loop item (this is the sequence item that you have just created, when you appended the loop item as described under 'creating trial blocks' above). Now append a form_text_input item. Rename it to 'open_question' and change the Form title property to 'Question:'.

Next up: magic. In the properties window, directly under where it says 'Your question', you could type a question. But you are going to refer to the variable 'q' you have created in the previous paragraph ('creating trial blocks'). You can do so, by changing the standard text ("Your question") to '[q]' (without the quotes, but with the square brackets!). Then press Enter and add the text "(Press Enter to confirm your answer)" on a newline. This way, participant's will know that when they need to press Enter to confirm their answer after they have typed it in.

Now run your experiment. Notice what happened? The question changed on each trial, because OpenSesame recognized the brackets and inferred that 'question' must be a variable.

This is great, but now you're stuck with having only open questions. To mend this, append a form_multiple_choice to your loop's sequence. Rename it 'multiple_choice', set the Form title to “Choose the option that matches your feelings best”, untick the box labeled 'Allow multiple options to be selected' and the box labeled 'Advance immediately to the next item once a selection has been made' (these labels speak for themselves regarding their functionality), and finally change “Your question” to '[q]' (again, without the quotes, but with the square brackets). All that is left to do now, is to create appropriate response options. Recall the first question? It was “Do you like bunnies?”, to which the following replies make sense: “Yes”, “No”, “Don't know” and “Only as food”. Enter these (each on a different line) in the box above which it says “Response options (different options on different lines)”.

Run your experiment again. You will have undoubtedly noticed that something is amiss. All of your questions have appeared twice, whereas you would like to show them just once. Of course, there is a simple way to fix this. In the overview, click on your loop's sequence. It should look like figure 4. As you can see in the highlighted part of figure four, OpenSesame gives you the option of using conditional statements[2] to run certain items (or not). The current *Run if* statement for open_question is set to 'always', meaning the item will run no matter what (which probably did not have to be pointed out; see how OpenSesame's interface is quite clear, even to new users?).



Flush pending key presses at sequence start: ✓

open_question	Run if: always
multiple_choice	Run if: always

Append existing item: open_question
Append new item: loop

Figure 4 – *Overview and sequence of your experiment up until now (Run if statement highlighted)*

Please look at figure 3 once more. Notice the variable 'qtype'? You can use this variable to create a conditional statement to use in the *Run if* statement of both items in your loop's sequence (i.e. open_question and multiple_choice). You can do so by typing in the following for open_question:

Run if [qtype] = 'open'

2 A 'conditional statement' sounds like rather difficult programming terminology, but it basically is a statement that can be either true of false.

The square brackets around 'qtype' tell OpenSesame that 'qtype' is a variable and that it should look at it's value for this cycle. The next part (= 'open') tells OpenSesame to compare the value for 'qtype' with 'open'. If they are the same, the open_question item is run. Otherwise, it is not. The same logic applies to multiple_choice, where the following should be entered:

Run if	[qtype] = 'mc'

Now give your experiment another run. If you did everything correctly, no double questions should appear this time. Your questionnaire now functions perfectly.

TIP! *Run if* statements may be used in a more complicated fashion. You can use '<' (smaller than), '>' (greater than) and even 'and' and 'or' operators to create elaborate statements.

logging data

Or does your experiment run perfectly? If you look at the output file, you'll see that it is completely empty! OpenSesame does not store anything in the data file, unless you ask for it. You can do so, by adding a logger item to your loop's sequence. A logger item stores the values of variables every time it is run. This means that adding a logger to the end of your trial sequence, will result in a logfile with the same number of row as the amount of trials that were run. A logger placed at the end of your experiment's sequence (*outside* of your trial sequence) will result in a logfile with a single row, containing the values of variables as they were at the end of the experiment.

What you want to do, is appending a logger to your loop's sequence. Place it at the end (underneath 'open_question' and 'multiple_choice'). If you click on your new logger item in the overview, you should see it's properties. Per default, a logger logs everything. This is mostly more data than you need, so it might be a good idea to cut down on this a bit (but do remember than when you forget to log something you need and run 300 participants; you risk acute major depression). The only variables you need to log for this experiment, are 'response', 'qtype', 'qnr' and possibly 'q'. You can do so by unticking the box labeled “Automatically detect and log all variables” in the logger's properties window. Next, select the variables you want to log (again, in the logger's properties window).

final run

Now run you experiment once more. After you finish, open the data file. OpenSesame's data files have a .csv file extension, because they contain comma separated values. This is quite a common data format, that can be imported into spreadsheet editors and SPSS directly. If you open the logfile with a spreadsheet editor, sometimes it is necessary to manually put the text to columns. In Excel and OpenOffice Calc, you can do so by selecting the entire A column, then clicking on 'Data' in the menu and selecting 'Text to columns'. A wizard will open, in which you should set the delimiter to a comma and the text separator to double quotes.

Experiment 2: Posner Cueing Task

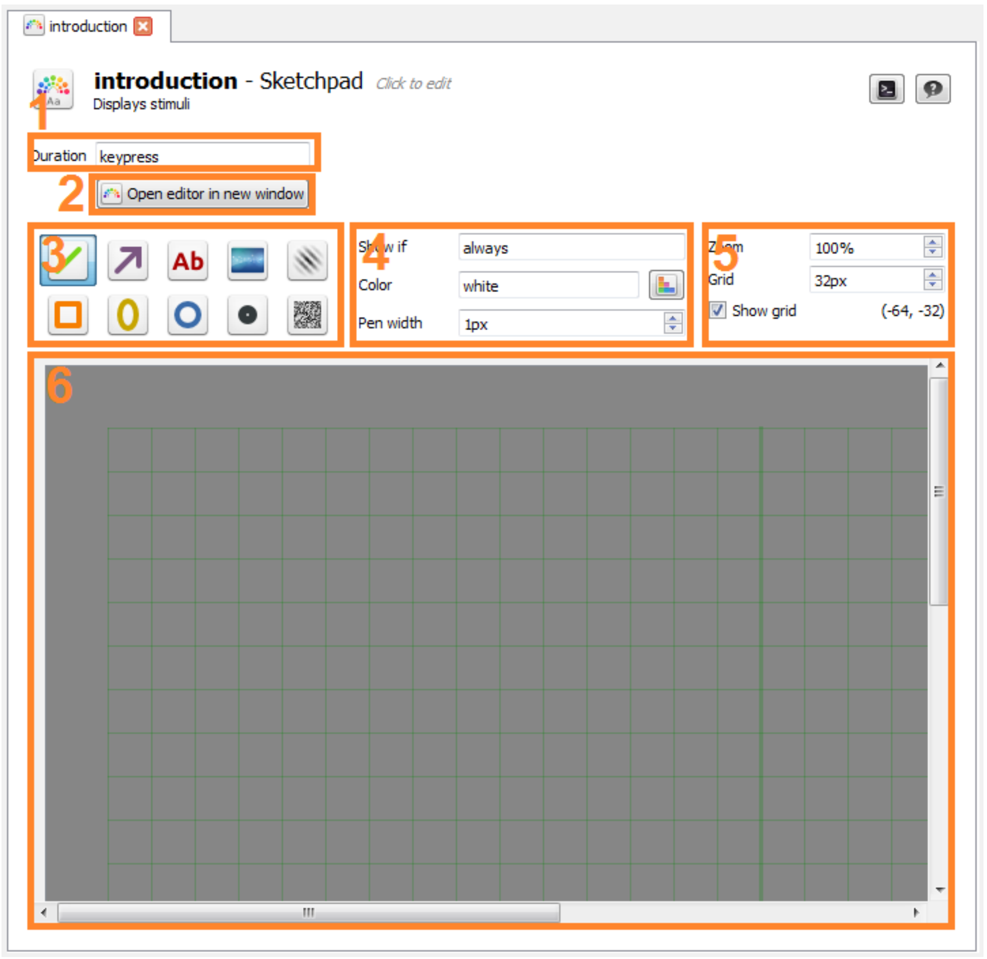
Now that you know your way around the GUI, it is time to create an actual experiment: a Posner Cueing Task (similar to Smith, Rorden, & Jackson, 2004). This will require some messing about with the script editor, and you will also learn to attach images to your experiment and present these.

getting started

Start OpenSesame, choose the default template and delete the 'getting_started' and 'welcome' items. Set the background colour to 'grey' and the foreground colour to 'black'. Please set the back-end to 'psycho'.

introduction

Append a sketchpad to the experiment sequence and rename the sketchpad to 'introduction'. An image ('introduction.png') should have been provided along with this document. This image contains the instructions for the Posner cuing task that you will be creating today (including fancy pictures!). You can add the image to your experiment, by clicking on 'View' (in the menu) and then on 'Show file pool'. Now click on the plus sign, browse to picture's location and double-click on the picture. The file should now be visible in the file pool.


introduction
introduction - Sketchpad *Click to edit*
Displays stimuli
1
2
Open editor in new window
3
4
5
6

Duration: keypress
Show if: always
Color: white
Pen width: 1px
Zoom: 100%
Grid: 32px
☑ Show grid (-64, -32)

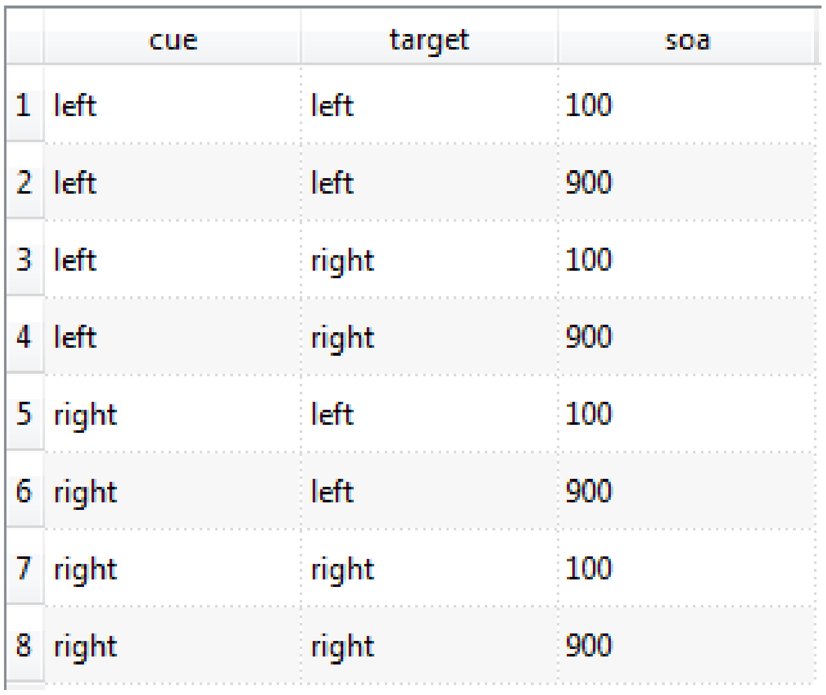
Figure 5 – *Properties window of a sketchpad item (see text for numbers and highlighting explanation)*

As is mentioned earlier, sketchpads can be used for all static visual stimuli. The properties window of a sketchpad is displayed in figure 5. Highlighted are:

1. Duration	Either a value in milliseconds, 'keypress' or 'mouseclick'
2. New window	Click this button to open the editor in a new window. This will give a better overview of your sketchpad.
3. Tools	These are all the tools available for creating stimuli on your sketchpad. From left to right, top to bottom: **Line tool**: click once to mark the begin point of the line, click a second time to mark it's ending. **Arrow tool**: works the same as the line tool, but creates an arrow instead. **Text tool**: click anywhere on the sketchpad. A window will open in which you can write the text you want to display. **Image tool**: click anywhere on the sketchpad. A window will open in which you can select any image in your file pool to display. **Gabor patch tool**: click anywhere on the sketchpad. A window will open in which you will be able to specify all of the Gabor's properties. **Rectangle tool**: click on the sketchpad once to mark the top left corner of the rectangle, click a second time to mark the bottom right corner. **Ellipse tool**: works the same as the rectangle tool, in the sense that you can specify the rectangle in the confinement of which the largest possible ellipse will be drawn. **Circle tool**: works a tad different from the ellipse tool (if it would do the same, having two different tools would make no sense). Click once to mark the circle's center, click a second time to indicate the circle's radius. A circle will be drawn based on those two properties. **Fixation dot tool**: click anywhere on the sketchpad to draw a fixation dot around that point. **Noise patch tool:** works much like the Gabor patch tool: click anywhere on the sketchpad and a window will open in which you can specify all of the noise patch's properties.
4. Tool options	After clicking on a tool, you can adjust some of the options (e.g. the colour, the penwidth and a *Show if* statement) for that tool here.
5. Grid options	The options for the grid displayed over the sketchpad are specified here.
6. Sketchpad	This is the drawing area of the sketchpad. Here you can create stimuli by using the tools mentioned at 3. Please note that the grid displayed here will not be visible to a participant. It is merely a tool to help you draw accurately.

To add the image to your experiment, click on the 'introduction' sketchpad. Now click on the image tool and then click on the center of the grid (if it helps, you can open the editor in a new window). After you have clicked in the sketchpad, a window opens and you can select a file from your file pool. Double click on 'introduction1.png' and it should appear in the center of your sketchpad.

single_trial
Append a loop item to the experiment sequence, and create a sequence to run with this loop when prompted. Please click on the loop's new sequence and rename it 'trial_sequence'. Time to fill out your loop. There is a need for three variables: cue (either 'right' or 'left'), target (either 'right' or 'left') and 'soa' (stimulus onset asynchrony; either 100 ms or 900 ms: you want to measure both validity effects and inhibition of return). Between these cues, there are (2 * 2 * 2 =) 8 unique trials. Therefore, set the number of cycles to 8 and complete the loop (an example is provided in figure 6).



	cue	target	soa
1	left	left	100
2	left	left	900
3	left	right	100
4	left	right	900
5	right	left	100
6	right	left	900
7	right	right	100
8	right	right	900

Figure 6 – *Completed Posner loop*

The observant reader will have figured out that there are no catch trials (i.e. target absent trials) in the current design. There is no particular reason to leave these out, other than for the sake of simplicity. If you feel outraged by this, please do not hesitate to add them yourself. In any case, the trial sequence for a single trial is summed up in table 1.

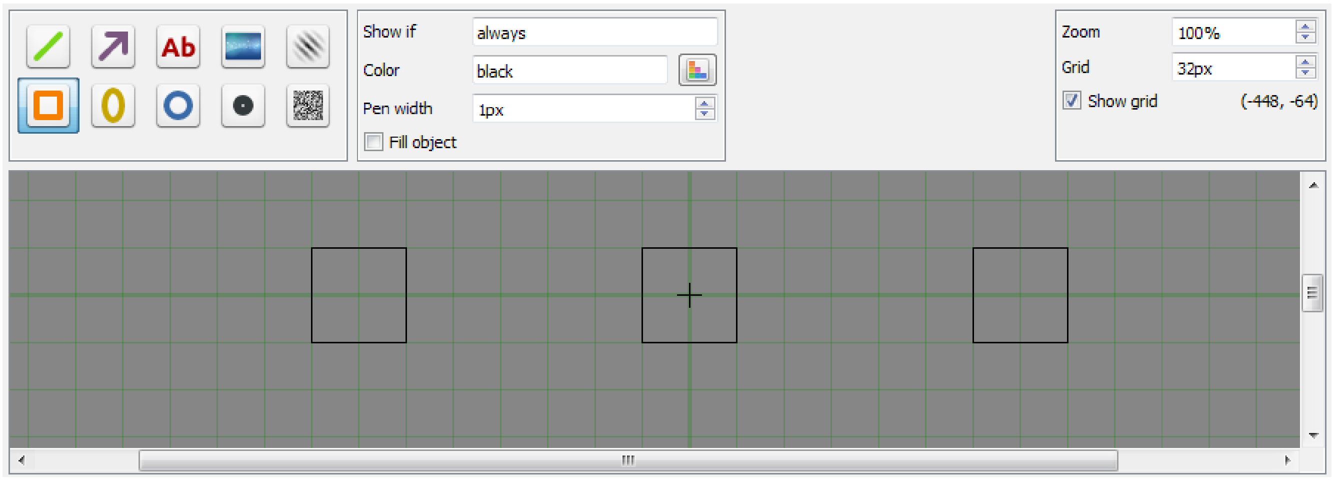
Table 1
Display sequence for a typical trial in the current Posner cueing task

screen	duration
fixation	1000 ms
cue	100 ms
SOA	100 or 900 ms
target	Until response

Not covered in table 1 is the temporal jitter you should add to the duration of the fixation screens, since you do not want your participant to anticipate the stimuli. In OpenSesame, introducing jitter is no big deal: you can use an 'advanced_delay' item and simply specify the values of your choice. More on this later. First, you are going to draw some sketchpads!

fixation

The fixation sketchpad should contain three horizontal boxes, of which the centre one contains a fixation cross (see figure 7). The borders of the boxes are 1 px wide, as are the lines of the fixation cross. The boxes are 64x64 px, the fixation cross' lines are 16 px long.



Show if: always
Color: black
Pen width: 1px
Fill object
Zoom: 100%
Grid: 32px
Show grid (-448, -64)

Figure 7 – *Fixation sketchpad*

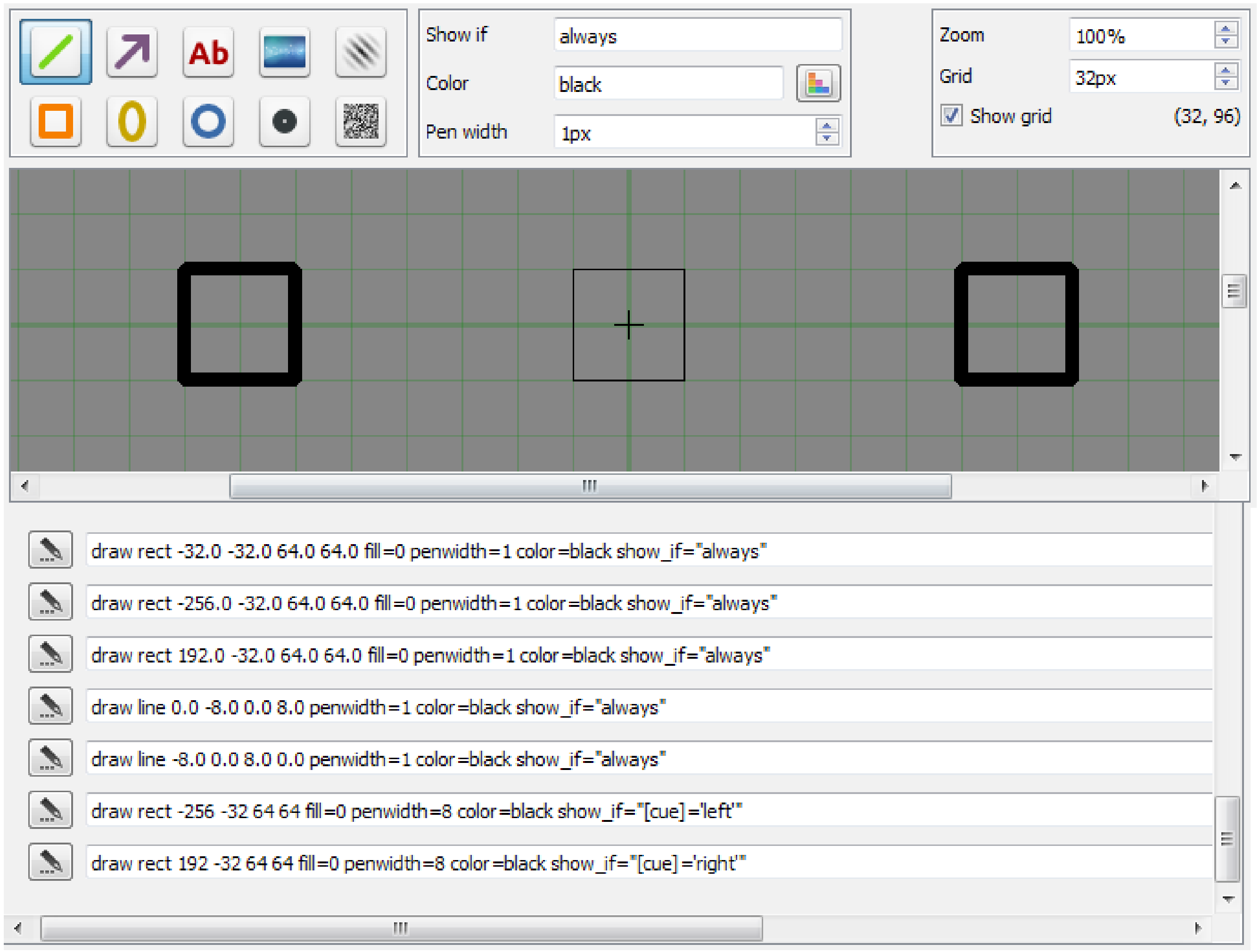
Set the Duration of the fixation sketchpad to 0 ms. After the fixation, append an advanced_delay and rename it fixation_delay. In the advanced_delay's properties, you can set the Duration to 1000 ms and the Jitter to 250 ms. If the Jitter mode is set to 'Std. Dev.', the delay is chosen from a normal distribution with M=Duration and SD=Jitter. If the Jitter mode is set to 'Uniform', the delay value is a random number between Duration–Jitter/2 and Duration+Jitter/2, which seems like the most appropriate option in this experiment.

The fact that you have set the fixation sketchpad's Duration to 0 ms, does not mean the sketchpad is hardly presented. It means OpenSesame will present your sketchpad and will immediately go on to the next item, which is an advanced_delay. The advanced_delay will not do anything with the display, but will simply wait for a bit while your fixation sketchpad remains on-screen.

cue

Append another sketchpad to trial_sequence. Rename it 'cue' and set the Duration to 100 ms. Adjust your sketchpad so, that it resembles figure 8. The cue should be a box with the same size as the regular boxes, but with a width of 8 px. The location of the cue should be dependent of the variable 'cue' you have created earlier (see figure 6). Of course, you cannot draw just one cue in the sketchpad; you have to draw two and tell OpenSesame that only one should actually be shown in the experiment. Remember the conditional statement you have encountered earlier (on pages 8 and 9), with the *Run if* statements for separate items? The same logic can be applied within a single sketchpad, to draw some stimuli, but leave out others.

Basically, you should simply copy the fixation screen, then draw two boxes with an 8 px penwidth over the left and right boxes. After drawing, press the little black button (labeled '>_') on the top right of the sketchpad's properties window. Clicking it will open the script editor. Although OpenSesame is Python-based, the script you see now is far from Python. It is OpenSesame syntax, that OpenSesame reverts to Python behind the scenes. A big advantage of this, is that the OpenSesame syntax has a very simple notation. Even if you are not code-savvy, you should be able to make (a bit of) sense of the script displayed on the next page. For more information on the OpenSesame script syntax, see here.


Show if: always
Color: black
Pen width: 1px
Zoom: 100%
Grid: 32px
[✓] Show grid (32, 96)

```
draw rect -32.0 -32.0 64.0 64.0 fill=0 penwidth=1 color=black show_if="always"
draw rect -256.0 -32.0 64.0 64.0 fill=0 penwidth=1 color=black show_if="always"
draw rect 192.0 -32.0 64.0 64.0 fill=0 penwidth=1 color=black show_if="always"
draw line 0.0 -8.0 0.0 8.0 penwidth=1 color=black show_if="always"
draw line -8.0 0.0 8.0 0.0 penwidth=1 color=black show_if="always"
draw rect -256 -32 64 64 fill=0 penwidth=8 color=black show_if="[cue]='left'"
draw rect 192 -32 64 64 fill=0 penwidth=8 color=black show_if="[cue]='right'"
```

Figure 8 – *Cue sketchpad*

If you have done everything correctly, you should see the following:

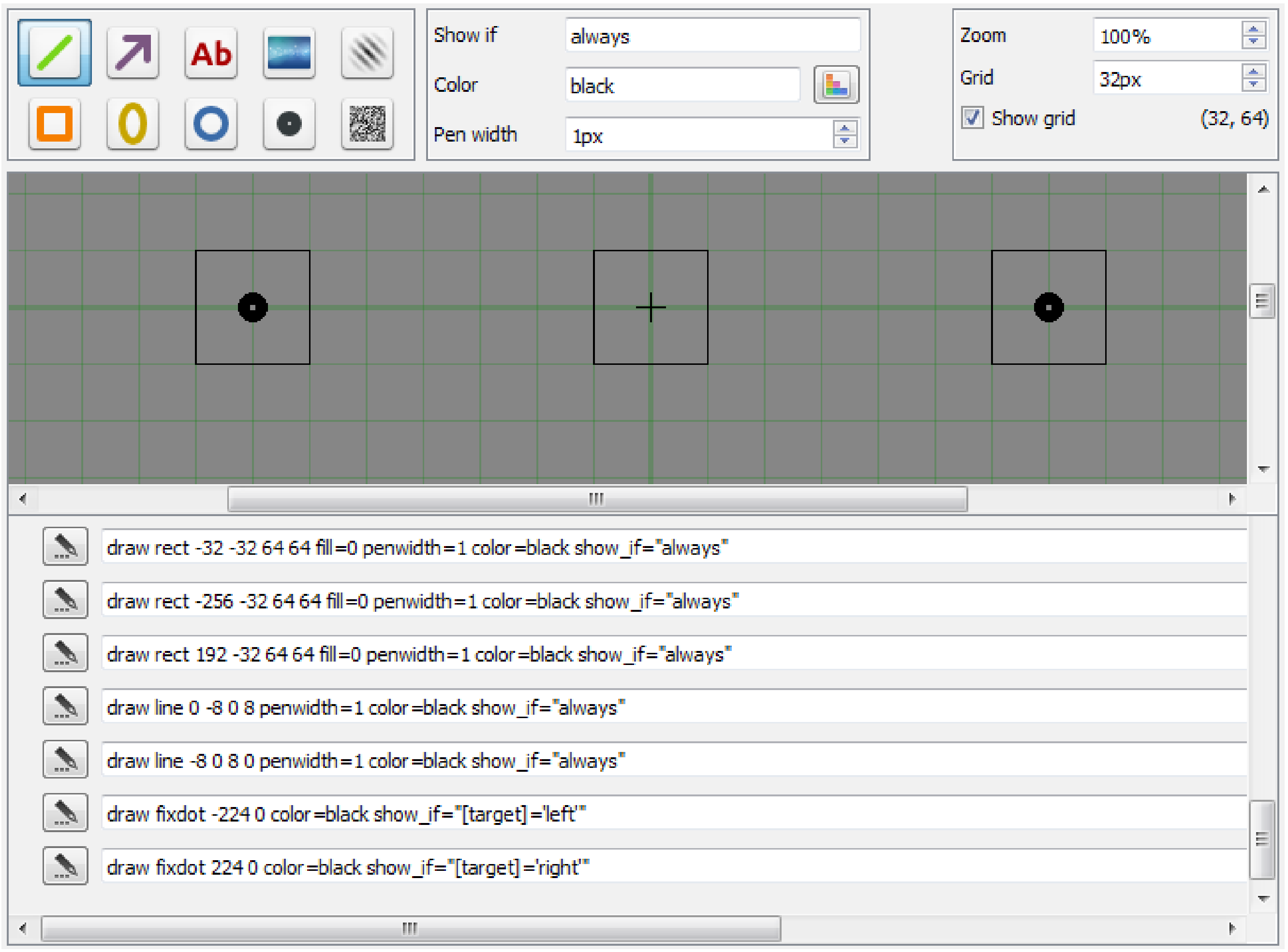
```
set duration "100"
set description "Displays stimuli"
draw rect -32 -32 64 64 fill=0 penwidth=1 color=black show_if="always"
draw rect -256 -32 64 64 fill=0 penwidth=1 color=black show_if="always"
draw rect 192 -32 64 64 fill=0 penwidth=1 color=black show_if="always"
draw line 0 -8 0 8 penwidth=1 color=black show_if="always"
draw line -8 0 8 0 penwidth=1 color=black show_if="always"
draw rect -256 -32 64 64 fill=0 penwidth=8 color=black show_if="always"
draw rect 192 -32 64 64 fill=0 penwidth=8 color=black show_if="always"
```

In OpenSesame syntax, you can use the square brackets to refer to variables, so you can change the last two lines to:

```
draw rect -256 -32 64 64 fill=0 penwidth=8 color=black show_if="[cue]='left'"
draw rect 192 -32 64 64 fill=0 penwidth=8 color=black show_if="[cue]='right'"
```

soa
Append yet another sketchpad and rename it 'soa'. This one should contain the same elements as the fixation screen. A sneaky way to accomplish this, is to copy the fixation sketchpad's script and paste it in the soa sketchpad's script editor. After doing so (and pressing 'Apply and close' to exit the script editor), the only thing you have to change, is the soa sketchpad's Duration. This should be set to '[soa]', as you could have figured out by yourself by now.

target
The final sketchpad to append, should be renamed to 'target'. It should look like the fixation sketchpad, but with targets in the center of the boxes on the left and right. These targets may be fixation dots, for example, as is demonstrated in figure 9. To make sure the target only appears on one side per trial, apply the same logic as with the cues (i.e. `show_if="[target]='right'"`).



Show if: always
Color: black
Pen width: 1px
Zoom: 100%
Grid: 32px
☑ Show grid

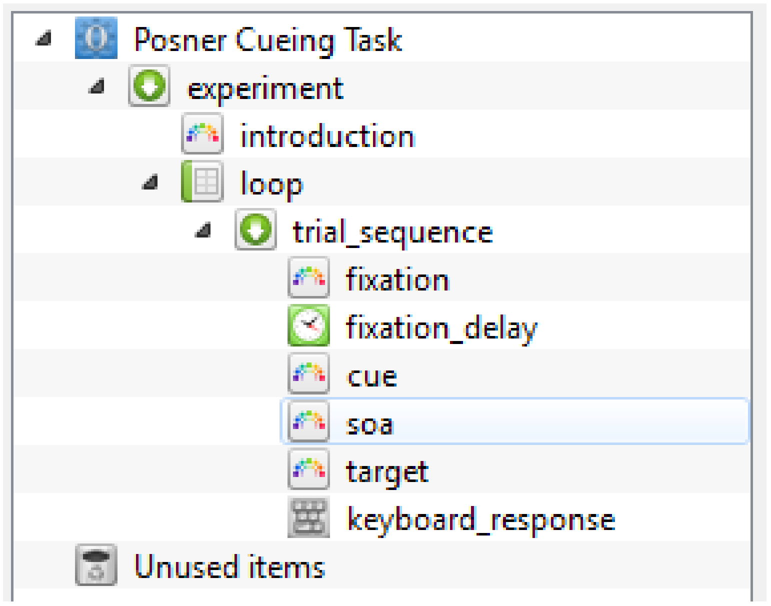
```
draw rect -32 -32 64 64 fill=0 penwidth=1 color=black show_if="always"
draw rect -256 -32 64 64 fill=0 penwidth=1 color=black show_if="always"
draw rect 192 -32 64 64 fill=0 penwidth=1 color=black show_if="always"
draw line 0 -8 0 8 penwidth=1 color=black show_if="always"
draw line -8 0 8 0 penwidth=1 color=black show_if="always"
draw fixdot -224 0 color=black show_if="[target]='left'"
draw fixdot 224 0 color=black show_if="[target]='right'"
```

Figure 9 – *Target sketchpad*

The target sketchpad's Duration may be set to 0 ms, as we are going to add a keyboard_response right after it (during which the screen is left unchanged, so the target will remain on-screen).

keyboard_response
Go back to trial_sequence and append a keyboard_response. Click it to check it's properties. The most important things to note, are the 'Correct response', the 'Allowed responses' and the 'Timeout'. The Timeout can be a value in milliseconds, or 'infinite'. The Allowed responses should be set to the keys you want participants to be able to use for this keyboard_response, separated by semicolons. In the current experiment, the right and left arrow keys should do the trick. In OpenSesame, these are called 'left' and 'right', so for the allowed response you enter: 'left;right' (without the quotes). Feel free to use other keys if you want: By clicking on the 'List available keys' button in the keyboard_response's properties window, you will be presented a list of all the available keys and their names. The Correct response differs per trial, as this is the target's location. Luckily, you have coded the target's location with 'left' and 'right', which happen to be the keynames of the appropriate responses. Therefore, you could enter '[target]' for the Correct response (unless you were stubborn and used other keys).

testrun
Now may be a good point to take a little brake from adding new stuff to your experiment and checking if your experiment works (hint: the overview should look like figure 10). You can test whether this is the case, by running your experiment.



```
Posner Cueing Task
    experiment
        introduction
        loop
            trial_sequence
                fixation
                fixation_delay
                cue
                soa
                target
                keyboard_response
Unused items
```

Figure 10 – *The overview as it should be by now*

finishing touch
Now, the only thing that's left to do, is to append a logger item to your trial_sequence and set the loop's Repeats to a decent number (about 20, or so). You could add a variable 'validity' that you could set to 'valid' if the target and cue are the same, or to 'invalid' if target and cue are different. This is something that could aid you in the analysis. Of course the validity can be determined afterward as well, by applying the same logic: if cue equals target, then the trial is valid; otherwise it is invalid.

Experiment 3: Using Inline Script and the Eyelink's Dummy Mode

The final experiment you will create in this workshop, is an experiment that you could conduct using an Eyelink (SR Research) eye tracker. Of course, these are hard to come by, but there is a workaround: OpenSesame does not only provide ready-made items for conducting eye tracking experiments (for Eyelink and SMI, and both in one set of plug-ins), it also gives you an extensive dummy mode to test those experiments without having to use the actual eye tracker. The current experiment will be a simple distractor task, in which participants will have to make an eye movement to a circle, while ignoring a square.

getting started
Start OpenSesame, choose the default template and delete the 'getting_started' and 'welcome' items. Set the background colour to 'grey' and the foreground colour to 'black'. Please set the back-end to 'legacy'. *Be sure to set the resolution to 1920x1080*!

introduction
Append a text_display, rename it 'introduction' and type whatever you would want the participant to know up front.

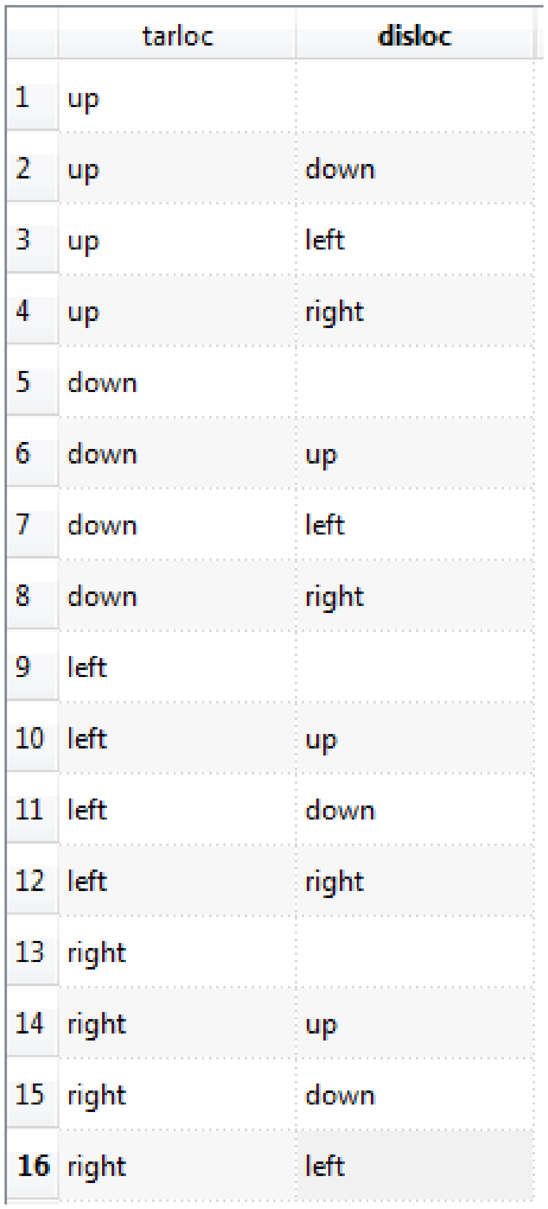
eyelink_calibrate
Append an eyelink_calibrate item and make sure to set it to dummy mode. You can do so, by setting the Tracker attached property to 'No (dummy mode)'. Normally, this item would set up the connection to the Eyelink PC and take care of the eye tracker calibration, but in dummy mode it simply sets up the dummy tracker (all behind the screens; this will not be visible for participants). No calibration is done, since the mouse is used as a simulator for eye movements and does not need calibrating.

loop
Next, append a loop and create a new sequence when prompted. Please rename the new sequence 'trial_sequence'. The loop should look like figure 11.

sketchpad
Append a sketchpad to trial_sequence. You can use the script below to create the stimulus screen:

```
set duration "0"
set description "Displays stimuli"
draw circle 0 -270 64 fill=1 penwidth=1 color=black show_if="[tarloc]='up'"
draw circle -270 0 64 fill=1 penwidth=1 color=black show_if="[tarloc]='left'"
draw circle 270 0 64 fill=1 penwidth=1 color=black show_if="[tarloc]='right'"
draw circle 0 270 64 fill=1 penwidth=1 color=black show_if="[tarloc]='down'"
draw rect -32 -302 64 64 fill=1 penwidth=1 color=black show_if="[disloc]='up'"
draw rect -302 -32 64 64 fill=1 penwidth=1 color=black show_if="[disloc]='left'"
draw rect 238 -32 64 64 fill=1 penwidth=1 color=black show_if="[disloc]='right'"
draw rect -32 238 64 64 fill=1 penwidth=1 color=black show_if="[disloc]='down'"
```

To implement it, go to your sketchpad and press the 'Edit script' button (labeled '>_'), select and delete everything and copy-paste the code above to the script editor. Press 'Apply and close' and you should be finished.



	tarloc	disloc
1	up	
2	up	down
3	up	left
4	up	right
5	down	
6	down	up
7	down	left
8	down	right
9	left	
10	left	up
11	left	down
12	left	right
13	right	
14	right	up
15	right	down
16	right	left

Figure 11 – *The loop as it should be by now*

eyelink_start_recording
Append an eyelink_start_recording item to trial_sequence, and place it before the sketchpad (you can do so by dragging and dropping the four black squares in front of the item's names in a sequence's properties window). This item will send a signal to the eye tracker to start recording when it would have been attached. In dummy mode, it makes sure the mouse is visible. Due to a bug in PyGame, the mouse will remain invisible on some Windows PCs. This is not common, and seems to happen primarily during demonstrations and workshops.

eyelink_wait
Next, append an eyelink_wait item to trial_sequence. You want the trial to run until the participant has made an eye movement, so you can set the eyelink_wait's Event property to "Saccade end".

eyelink_stop_recording
Append an eyelink_stop_recording item. This will tell the eyelink to stop recording when an eye tracker is attached. In dummy mode, the mouse is hidden again.

pause
Finally, append a sketchpad, rename it 'pause', and draw a fixation dot in the center. You can set the Duration to a second, as this is an inter trial pause for the participant.

testrun
Curious if this works? Please help yourself and run the experiment!

inline_script
The experiment is fine as it is, which is kind of amazing, considering the amount of work a programmer would normally have to put in creating a similar experiment in a programming language. That being said, it would be nice if the participant received some feedback. Also, you want the participant to respond as quickly as possible, in hopes of a larger attentional capture rate. Therefore, it would be great if we could see where a participant's saccade ended and compare this ending point with the stimulus location. Of course, showing some feedback after a tardy response would be great as well. We can do both of these things, by using an inline_script item.

An inline_script item van be used to run a bit of Python code. This is fundamentally different from the OpenSesame syntaxt you have seen before, as Python is an actual programming language. More information on Python inline coding and learning Python can be found here; the specific Python inline functions of the eyelink plugins can be found here.

Delete the eyelink_wait item from trial_sequence. Then, append an inline_script item and place it where the eyelink_wait used to be (between the sketchpad item that presents the stimuli and the eyelink_stop_recording). Now in the inline script, type the following:

```
# get beginning timestamp
t0 = self.time()
```

Python disregards everything behind the number sign (that is why the text turns to green), so you can use it to write comments (so that, later on, you still know what your script was for). On the next line, you have used the function `self.time()`, which returns the time since the beginning of the experiment. This value is stored in the variable named `t0`. On to the next bit of scripting:

```
# wait for saccade
t1, startpos, endpos = exp.eyelink.wait_for_saccade_end()
```

Again, you see the commenting. Then, a slightly more difficult expression is used. When you analyze it carefully, you see that three variables are created: `t1`, `startpos` and `endpos`. These variables are given their values by the function `exp.eyelink.wait_for_saccade_end()`, which simply waits until a participant has made a saccade.

After a participant has made an eye movement, you should be able to calculate the response time:

```
# calculate response time
resptime = t1 - t0
```

Since `t1` contains the timestamp of when the participant's saccade ended and `t0` contains the saccade's starting timestamp, `t1` minus `t0` should be the time passed in between (in which the participant made an eye movement). Now, let's check if the response time is to our liking:

```
# check if response time is ok
if resptime < 800:
      faster = 0
else:
      faster = 1
```

What you see here, is an if-statement. It checks if the response time was below 800 (ms). If this is true, the variable `faster` is set to `0`. In all other cases (i.e.: if the response time was greater then 800), the variable `faster` is set to `1`. Later on, we can use this variable to check whether feedback on the participant's response time has to be given or not.

```
# check target location
if self.get("tarloc") == "up":
      tarcor = (960,270)
elif self.get("tarloc") == "down":
      tarcor = (960,810)
elif self.get("tarloc") == "left":
      tarcor = (690,540)
elif self.get("tarloc") == "right":
      tarcor = (1230,540)
```

The next piece of inline script determines the centre coordinate of the target. Again, this is done via an if-statement. To understand the code, it helps to know that `elif` is an abbreviation of 'else if'. Another new part in the code, is the use of the function `self.get(“variable_name”)` to get the value of the variable `tarloc`. Why not simply use `[tarloc]` or just `tarloc`? This is an important difference between OpenSesame syntax and Python scripting. Variables you define in OpenSesame's GUI are available in the OpenSesame syntax by using square brackets around the variable name. These square brackets, however, have a different meaning in Python. You cannot simply use the variable name either, as the variables defined in OpenSesame's GUI are not *locally* available in an inline_script item. This is why you need to explicitly get the variable's value, by using the `self.get` function.

The rest of the script (e.g. `tarcor = (960, 270)`) seems to make sense: these are simply coordinates for the target centres. You want to figure out if the saccade was made towards these or not. To do so, you should first calculate the distance from the saccade's ending position to the target coordinate.

```
# calculate if endpos matches target location
distance = ((endpos[0]-tarcor[0])**2 + (endpos[1]-tarcor[1])**2)**0.5
```

Although this is only one line of code, it seems horribly complicated. There is no need to panic, though, as this is just Pythagoras' theorem. It is written down in a slightly less straightforward way than the familiar $a^2 + b^2 = c^2$.

If you work from the inside out, you can see that first both the horizontal and the vertical distance between the ending position and the target's centre coordinate is calculated (horizontal: `endpos[0] – tarcor[0]`; vertical: `endpos[1] – tarcor[1]`). The expression `tarcor[0]` means 'the first position of variable `tarcor`, and `tarcor[1]` means 'the second position of `tarcor`' (important note: in Python, counting starts at 0). When the horizontal distance (a) and the vertical distance (b) are calculated, they are both squared (`**2`) and then summed.

To recap: you now have $a^2 + b^2$, which means you know c^2 (since $a^2 + b^2 = c^2$). The only thing you need to do now, is calculate the square root of c^2 to know c, which is the distance between the saccade's ending position and the target coordinate. Since taking the square root of a number is the same as exponentiating the same value with a half, you use `**0.5` to calculate the square root of c^2.

```
# check if distance between target and saccade endpoint is ok
if distance < 64:
	correct = 1
else:
	correct = 0
```

You should be able to see right through this bit of code: it checks whether the distance is smaller than 64, which is twice target circle's radius, to catch saccades that land on or slightly off target. If the distance is below 64, the variable `correct` is set to `1`, otherwise it is set to `0`.

Since you now know everything you set out to find out (i.e. the response time and the correctness of the response), you are almost finished with your inline_script. Almost, since there is still a small (but very significant!) detail to attend to. Just now (on page 20), you have learned that variables defined in OpenSesame's GUI are not locally available in an inline_script item. The exact opposite is true as well: variables that are used locally in an inline_script item, are not available in OpenSesame's GUI, unless you explicitly make them available. The following piece of code does this:

```
# make some variables available for OpenSesame
exp.set("correct", correct)
exp.set("faster", faster)
exp.set("response_time", resptime)
```

The function `exp.set` uses the first argument (e.g. `"response_time"`) to create a variable with that name, which is available in OpenSesame's GUI. Next, it gives it the value of the second argument (e.g. `resptime`). So, if a participant responded in 548 ms, but looked at the wrong place on the screen, the inline_script would set the variable [correct] to 0, the variable [faster] to 0, and the variable [response_time] to 548. All three of these variables would be usable in OpenSesame's GUI.

feedback
Since you now have some variables to use for giving feedback, why not do so? Append a feedback item to trial_sequence. Place the feedback item between eyelink_stop_recording and pause (see figure 12). A feedback item has the same functionality as a sketchpad item, so you should know how to write some text in it. You should give some feedback (e.g. “That is WRONG!”) if [correct]=0 and some feedback (e.g. “YOU'RE TO SLOW!”) if [faster]=1. Make sure that your feedback is in a very large font, and preferably in red, so that the participant knows that you are *very* serious about this! The following script should suffice:

```
set duration "500"
set reset_variables "yes"
set description "Provides feedback to the participant"
draw textline 0 -160 "INCORRECT!" center=1 color=red font_family="mono"
font_size=50 font_italic=no font_bold=no show_if="[correct]=0"
draw textline 0 160 "RESPOND FASTER!" center=1 color=red font_family="mono"
font_size=50 font_italic=no font_bold=no show_if="[faster]=1"
```

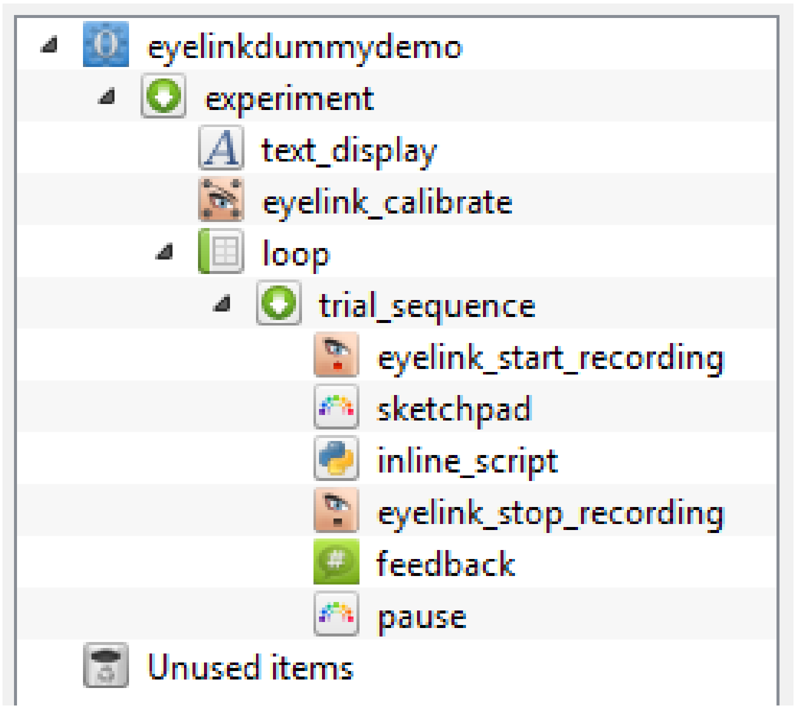


Figure 12 – *The overview as it should be by now*

Where to Look for Help?

Congratulations on your first (big!) steps into OpenSesame. You should be able to do quite a bit already, but it is very likely that you run into trouble along the way somewhere. This is to be expected, and you would most certainly not be the only one to come across difficulties when learning how to use a new piece of software.

One of the arguments a lot of researchers use against open-source software, is that it lacks decent support. While this holds true up to a certain extent, since some developers have abandoned their projects or just aren't into giving feedback, OpenSesame is not like that. There is an excellent documentation site with information on everything related to OpenSesame. There is a forum as well, on which users can ask questions, which are answered by other users and members of OpenSesame's development team.

Documentation Site osdoc.cogsci.nl

On the documentation site, you can say hi to the team, you can download every single version of OpenSesame there is (even an experimental version for Android), you can learn how to use OpenSesame, you can learn how to use Python in inline_scripts and find all inline_script functions (e.g. the canvas functions), you could master forms for a questionnaire, you can find a list of available plug-ins, delve into the back-ends, check out which external devices are compatible with OpenSesame (response boxes and other serial and parallel port devices, EEG equipment and audio input, to name a few), you can learn about timing and test your own system, about counterbalancing, about using non-western alphabets, you can find example experiments, a list of publications for which OpenSesame was used, you can find out how to contribute and you could donate to the project (but please do not feel obliged to do so: we love giving you our excellent experiment builder for free).

Support Forum forum.cogsci.nl

If you cannot find your answer on the documentation page, be sure to ask around on the forum. The forum is available to everyone, but to post your own questions, you are required to register. This is free, though, so no worries there. On the forum, Sebastiaan Mathôt (OpenSesame's lead developer) is very active, as are the other members of the OpenSesame team and an increasing number of users. If you post a question here, it is very likely that someone can solve your problem, help you out a great deal, or at least point you in the right direction. Be sure to search for existing topics first, though, as your question might have already been answered in the past.

This concludes our tutorial. I hope you have enjoyed getting to know OpenSesame, or at least have seen its use for your future studies. On behalf of the entire team, I wish you the best of luck on your research.

References

Brainard, D.H. (1997). The Psychophysics Toolbox. *Spatial Vision*, *10*(4), p. 433-436.

Halchenko, Y.O., & Hanke, M. (2012). Open is not enough. Let's take the next step: An integrated, community-driven computing platform for neuroscience. *Frontiers in Neuroinformatics*, *6*:22.

Mathot, Schreij, & Theeuwes (2012). OpenSesame: An open-source, graphical experiment builder for the social sciences. *Behavioral Research Methods*, *44(2)*, p. 314-324.

Pierce, J.W. (2007). PsychoPy – Psychophysics software in Python. *Journal of Neuroscience Methods*, *162*, p. 8-13.

Pierce, J.W. (2009). Generating stimuli for neuroscience using PsychoPy. *Frontiers in Neuroinformatics*, *2*:10.

Smith, D.T., Rorden, C., & Jackson, S.R. (2004). Exogenous Orienting of Attention Depends upon the Ability to Execute Eye Movements. *Current Biology*, *14*, p. 792-795.

Van Rossem, G., & Drake, F.L. (2011). Python language reference manual. Bristol, UK: Network Theory Limited.

APPENDIX A

List of Items and their Functions

Simple Stimulus Presentation

Visual

name	from	description
sketchpad	standard installation	Displays a large variety of visual stimuli, with built in tools for creating lines, arrows, rectangles, ellipses, circles, fixation dots, images, gabors, random noise patches and text.
fixation_dot	standard installation	Displays a central fixation dot or cross.
text_display	standard installation	Displays a text (sidenote: forms are faster and more flexible)
video_player[3]	cogsci.nl plugins	Shows video with sound and audio (somewhat deprecated).
media_player	github	Shows video with sound and audio.
media_player_vlc	github	Shows video with sound and audio using VLC Media Player.

Auditive

sampler	standard installation	Plays a sound file. Volume, pan (left-to-right sound distribution), pitch, fade-in, fade-out and duration can be adjusted.
synth	standard installation	A sound synthesizer to create sine, sawtooth or square waves or white noise, with adjustable attack (fade-in), decay (fade-out), volume, pan (left-to-right sound distribution) and duration.

3 Another way of playing videos without using a plugin, is using OpenCV, a computer vision library.

Form Stimulus Presentation (e.g.: questionnaire items)

form_base	standard installation[4]	Completely customizable form.
form_text_display	standard installation	Displays a text and a button to press to continue.
form_consent	standard installation	Displays an informed consent form with an 'I agree' option.
form_multiple_choice	standard installation	Displays a question with multiple answers.
form_text_input	standard installation	Displays a question and allows for a participant to type in text.

Response collection

keyboard_response	standard installation	Waits for a keyboard press (or a timeout). Allows for selecting pressable keys and identification of correct responses.
mouse_response	standard installation	Waits for a mouse button press (or a timeout). Allows for selecting pressable buttons and identification of correct responses.
joystick	standard installation	Waits for a joystick button press (or a timeout) and introduces joystick functions to be used in inline_script items
srbox	standard installation	Collects input from a Psychology Software Tools serial response box
port_reader	standard installation	Reads input from a serial or a parallel port (Windows only)
slider	github	Presents a question with a slider that participants can fill or unfill with the mouse
text_input	standard installation	Collects text input to a presented question (similar to form_text_input and somewhat deprecated)

4 From OpenSesame 0.27 and on, form plugins are included per default

Eyelink

eyelink_calibrate	Eyelink plugins	Sets up connection to an Eyelink and allows a user to calibrate the eye tracker using both the Eyelink or stimulus PC
eyelink_drift_correct	Eyelink plugins	Performs a drift check, allows a user to switch to calibration screen
eyelink_log	Eyelink plugins	Writes a message to the EDF file
eyelink_start_recording	Eyelink plugins	Starts Eyelink recording
eyelink_stop_recording	Eyelink plugins	Stops Eyelink recording
eyelink_wait	Eyelink plugins	Waits for a specified event

Miscellaneous

advanced_delay	standard installation	Waits for a specified amount of time; allows for the possibility to add jitter (either from a normally distributed curve or from a random distribution)
external_script	standard installation	Runs a Python script from an external file
notepad	standard installation	An item for adding comments; it has no effect on the experiment flow whatsoever
sequence	standard installation	An item that sequentially runs a number of other items
parallel	standard installation	An item that runs a number of other items parallel to each other
reset_feedback	standard installation	Resets the feedback variables (e.g. 'avg_rt' and 'acc')

Questionnaire[5]

text_screen	github	Presents a simple text screen
open_question	github	Presents a question and a text input field
multiple_choice	github	Shows a question and multiple response options with checkboxes
rating_scale	github	Provides a rating scale

5 Please note that the questionnaire plugins are not longer actively maintained and have been replaced by the form plugins