GRADE Workshop Frankfurt, May 11-12th 2021
Aylin Kallmayer, Melvin Kallmayer, Leah Kumle

Psychopy Tutorial – Gaze Cue Experiment

This Psychopy tutorial consists of two parts. In part one you will program a single trial sequence for a simple gaze cue experiment (see Figure 1 for trial sequence) as introduced by Friesen and Kingstone (1998). You will use the Psychopy coder to implement 4 functions that take care of presenting the four respective parts of the trial. In part two you will extend your code to present multiple trials in varying conditions.

Go through this Tutorial step-by-step and try solving each task on your own first. We will not always go through the script chronologically, but the tasks will always tell you exactly where in the script you're supposed to do something so don't be confused if you're working on a part but haven't worked on another part that's "above" your current task in the script. If you feel like you're stuck you can check the helper document, ask someone in your group for help, look up the Psychopy documentation online, or ask one of us for help. Link to Psychopy documentation: https://www.psychopy.org/api/index.html

Figure 1. Trial sequence gaze cue experiment. Correct response for target "F" is key "y", for target "H" is "m".

PART 1

1. First, please open Psychopy and go to the **cover view**, which should look something like this:

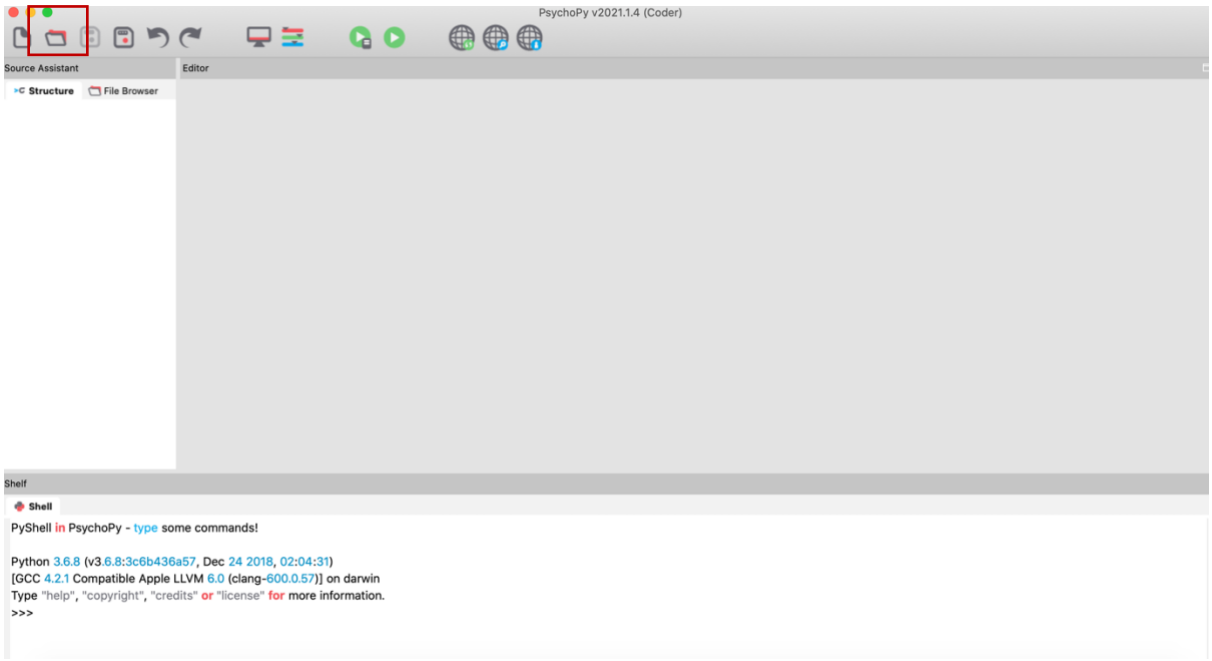

Figure 2. Psychopy Coder view.

By clicking on the icon highlighted by the red box you can open the file **gaze_cue_tut_part1.py** which is located in your workshop folder under day2:

Day2/Gaze_cue_experiment/gaze_cue_tut_part1.py

This should open a **script** which looks like a text file with different colors for different parts of the code.

A script is essentially just a text file of your code that can be executed. Python will "read" the script from top to bottom!

Right now, if you **executed the script** (by clicking the green play button) it would simply show you a grey screen with the words "END" for one second. The goal for the first part of this tutorial is for you to **integrate functions that take care of presenting the different screens that are part of the trial sequence** (Figure 1). But first, let's go through the first part of the script (you don't have to implement anything here, just try to understand what's going on)

a. **SET UP** – here we are doing 3 things:
    i. First, we are loading necessary **libraries** which we will need to program our experiment.
    ii. Whenever we create stimuli in Psychopy we will need to specify the window on which they should be drawn on. That's why here we are creating this **window** which we call `win`, defining its size, units, color and position.
    iii. To be able to quit the experiment at any given time we are creating an **"escape" option** assigned to the "q" key.
b. **DEFINE CONDITIONS** – Here we store specific information about the trial in a list which we call `conditions`. We want to program a single trial with the gaze cue in the condition "**left**", the target letter "**F**" and the target position also in the condition "**left**".

2. **DEFINE FUNCTIONS** – The first function that we define is called `one_trial(gaze, target, target_pos)`. The idea is, that this function takes our three variables **– gaze, target letter, target position –** that we have stored in `conditions` as input and handles a full trial sequence (Figure 1). Right now, this function is mostly empty except for the `end_screen()` part (which presents the grey "END" screen you saw before). Once you have finished all the steps of this first part of the tutorial, the `one_trial(gaze, target, target_pos)` function should take care of the following things:
    a. Presenting the fixation cross in the middle of the screen for 750ms
    b. Presenting the neutral gaze image in the middle of the screen for 750ms
    c. Presenting the gaze cue (left) in the middle of the screen for 500ms
    d. Presenting gaze cue in the middle, target letter (left) and distractor (right), waiting for a key response.

    Instead of defining all of the above steps in the `one_trial(gaze, target, target_pos)` function itself, we will write **individual functions** for each step (a-d) and combine them in the `one_trial(gaze, target, target_pos)` function.

Important: after finishing one of the functions you need to also call them in the one_trial(gaze, target, target_pos) function (similarly to the end_screen() function). That way, you can test your implementation after each step by running the script.

*Tip: When creating stimuli, most functions need you to define a window, in our case you should use the window we created under SETUP called **win**.*

1) **present_fix_cross()**

   Our trial begins with a fixation cross. This function should present a fixation cross in the middle of the screen then flip the window and wait for 750ms. Feel free to play around with the size of the fixation cross. Check the helper document for help. *Feel free to play around with the size of the fixation cross!*

2) **present_neutral_gaze()**

   This function presents the gaze_neutral.png image in the middle of the screen for 750ms. It works exactly like 1) but instead of a text stim you create an image stim. *If you need help defining the image path check the helper doc!*

Remember you can test out your code after each step to see your progress!

3) **present_gaze_cue(gaze)**

   Here, we present the gaze cue image in the middle of the screen for 500ms. It works exactly like 1) and 2). Notice how this function now has one input argument, **gaze**, which can be either "left" or "right". Later, this can be used to vary which gaze cue image should be presented here, but for now we only need this function to be able to show the left gaze cue image.
   If you need help defining the image path check the helper doc!

4) **make_response_screen(gaze, target, target_pos)**

   We will now build our response screen (gaze cue with target and distractor letter).

   a. For now, let's start by presenting the two letters on screen. Our target position is **left** and our target letter is **F**.

   Important: Psychopy's `visual.TextStim` function doesn't accept "left" or

"right" as arguments for position. We need to translate them into **tuples** first (more on tuples later): In the function, create a variable called `target_tuple` and set it to (-200,0) and create another variable called `dist_tuple` which you set to (200,0). Now you can use these two variables in the `visual.TextStim` function to assign the positions for target and distractor letters. Then flip the window and wait for one second.
Go ahead and include `make_response_screen(gaze, target, target_pos)` in the `one_trial(gaze, target, target_pos)` now and test it out. You should see the letter **F** on the left side and the letter **X** on the right side.

b. Now all that's left to do is adding the gaze cue from 3) meaning you need to create the image path and `visual.ImageStim` again.

c. The task in this experiment is for the participant to press the correct key (depending on the target letter). However, our usual sequence of drawing the stimulus, flipping the window and waiting that we have applied so far:
`win.flip()`
`core.wait(1)`
does not allow us to wait for a response from the participant. Check this link https://www.psychopy.org/api/event.html to find a psychopy function that waits for a keypress before continuing (maybe also check the introduction to psychopy presentation from earlier…). Once you've found the correct function replace `core.wait(1)` with that function. Your function might have an argument called `keyList`. If you don't understand what that represents check the helper document again.

3. **RUN EXPERIMENT** – here, you don't need to edit anything. In the first line we are "unpacking" the conditions and giving them as input to our `one_trial(gaze, target, target_pos)` function.

Amazing!!! You just programmed the whole trial sequence! When running your script, it should show exactly what you see in Figure 1. If everything seems to be working you

have finished part 1 of this tutorial. Please let one of us know that you're done. Maybe you can help someone else in your group?

PART 2

To begin part two of this tutorial we will first increase the number of trials. This means, instead of calling the `one_trial(gaze, target, target_pos)` function once, we want to call it multiple times with a set number of repetitions.

4. Increasing the number of trials

To repeat our trial sequence multiple times, we can do something you already know how to do – looping! In your script (we're still in gaze_cue_tut_part1.py) under RUN EXPERIMENT "run multiple trials" write a loop, where for each iteration you run a single trial sequence so that in the end the trial is repeated 3 times. *(If you don't comment out the line below "run single trial" you will run 4 trials overall.)*

5. Vary task condition

As you can see, we are now showing the same trial three times without varying the **gaze**, **target** or **position** condition. If we want the participant to go through all possible combinations of our variables (not only left gaze cue but also right etc.) we need to change a few things. Now is the time to open the second script located in the same folder called **caze_cue_tut_part2.py.** There are a few things that are different now, let's go through them step by step.

a. Change the for loop

Let's start with our **conditions** list. It looks a bit different now! We now have **tuples** (round brackets ()) inside of a **list** (square brackets []). Tuples are similar to lists as they are a collection of ordered items. The big difference is, **tuples cannot be modified after they are created**, i.e. you can't add anything to them or change the order of contained items! We will use this property to our advantage as the order of the items in the tuple (gaze condition, target letter condition, target letter position) **should not be changed or appended by accident**. The order of the tuples (remember each tuple represents one trial!) within the list

however is not really important which is why it's fine to use a list here -> [].
As mentioned, each tuple contains the conditions for a single trial and we already included two types of trials for you.

If we now want to present these two trials, first with gaze cue left and then with gaze cue right, we can simply **loop through our conditions list** and for each iteration unpack the tuple to extract gaze, target letter and target position for that trial, then calling the `one_trial` function with current gaze, target letter and target position as input.

Go ahead and add a loop under RUN EXPERIMENT "run multiple trials" that presents the two trials defined in the `conditions` list.

b. Add all combinations to the conditions list

Now, we want to fill up our conditions list with all possible combinations of gaze, target letter and target position. Under DEFINE CONDITIONS edit `conditions` list so that it includes trials for all combinations of gaze cue, target letter and target position. Run the script again to test out your solution.

c. Edit make_response_screen(gaze, target, target_pos) function

If you payed really close attention you might have noticed that our target letter still appears only on the left, no matter whether you have target position "right" or "left". Can you come up with an explanation for this? Check the `make_response_screen(gaze, target, target_pos)` function again, do you notice anything?
We hardcoded the `target_tuple` and `dist_tuple` positions in the function! So, it doesn't matter whether we give the function "left" or "right" as the third argument, it will always compute `target_tuple = (-200,0).` If we want the target and distractor to appear on the left or right depending on the condition, we need to add an **if statement** here. Check the hints if you're not sure how to do this.

d. Randomization

Now, if we want to randomize the order of the trials, try googling how to randomize the order of items in a list and then put that statement under RUN EXPERIMENT "randomize items of list".

6. Data logging

a. To save participants responses we need to edit `event.waitKeys()` in our `make_response_screen(gaze, target, target_pos)` function. So far, psychopy simply waits for the key to be pressed but does not store the pressed key anywhere. Replace the `event.waitKeys()` line with the following:

```
response = event.waitKeys(keyList=["y","m"])[0]
```

What's happening here: `event.waitKeys` returns a list of keys that were pressed. So with the first index [0] we're accessing the first key pressed.

b. To finish this up, you need to **return** `response` in the `make_response_screen(gaze, target, target_pos)` function as well as in the `one_trial(gaze, target, target_pos)` function, otherwise you won't have access to it outside of the function and won't be able to log responses anywhere.

To be able to return `response` in the `one_trial(gaze, target, target_pos)` function, we need to log the response first when `make_response_screen(gaze, target, target_pos)` is called within the `one_trial` function by assigning the output to a variable. We need to do the same when we later call the `one_trial(gaze, target, target_pos)` function under RUN EXPERIMENT.

Check the helper document if this was (understandably) a bit confusing!

c. To log responses into a text file, go back to SETUP "prepare output file". This is where we create an empty file and assign the column names. Notice we have a

column called “subject_id”. If you want this to log your name, edit the `subject_id` variable under SETUP “assign subject id”. To log responses after each trial, go into your loop under RUN EXPERIMENT “run multiple trials” and add this line (should be the final line within your loop).

```
writer.writerow([subject_id, gaze, target, target_pos, response])
```

If you assigned the output of `one_trial(gaze, target, target_pos)` to something other than response you will need to change it here aswell!

This “writes” the state of these variables for this trial into our textfile.
Go ahead and test this out by running the script and then checking the textfile created

7. **Bonus: accuracy**

   If you feel up for it you could now write your own function that for each trial checks whether the correct key was pressed. You can then use the function to store the accuracy of the keypress (e.g. 0 for wrong and 1 for correct) in the output file.

8. **You’re bored?**

   If you happen to be finished super early and still want more challenges be sure to approach one of us and we will give you some more tasks to do!

References

Friesen, C. K., & Kingstone, A. (1998). The eyes have it! Reflexive orienting is triggered by nonpredictive gaze. *Psychonomic Bulletin & Review*, *5*, 490–495. doi:10.3758/BF03208827