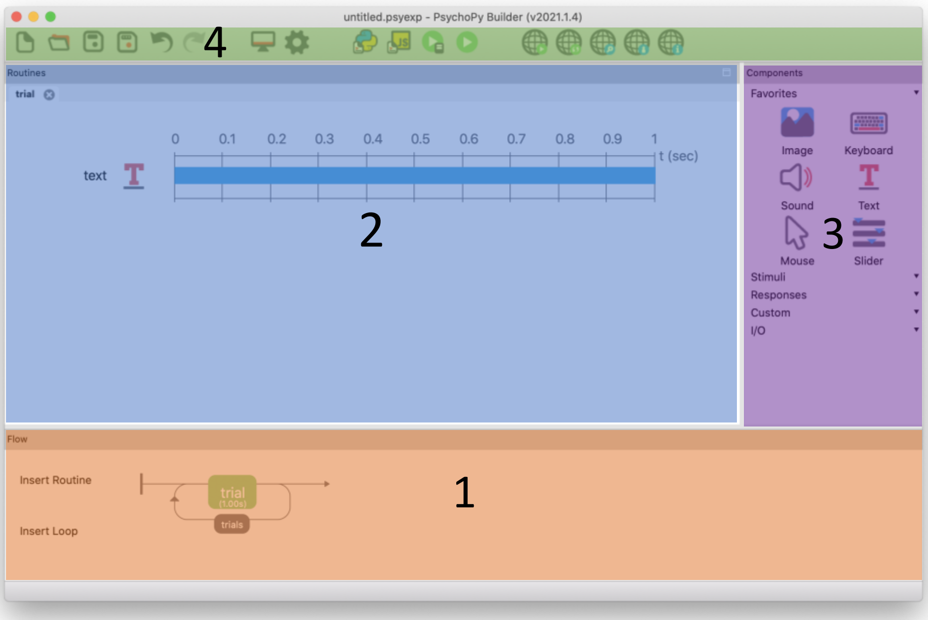
About This Tutorial

This tutorial will guide you step-by-step through building a visual search experiment in the PsychoPy Builder. We recommend working through this with your group and asking each other for help if you get stuck. It takes about 2 hours of work, and you can work on it in one go or do bits of it with breaks in between. Just remember to always save your progress! The idea is that by the end, you have learnt a bit about how to build experiments using the PsychoPy Builder and know how to implement some python code. **You don't need any prior experience with coding to work through this tutorial!** If you're struggling, ask other members in your group for help, use the PsychopPy forum (https://discourse.psychopy.org), google, or ask us in slack. Good luck and have fun!

PsychoPy Builder – Overview

Let's first have a look at the PsychoPy Builder layout. If you open PsychoPy in the Builder view, it should look something like this:



Let's walk through the different areas of the builder and what they do:

1. Flow: **Routines** and **Loops**
   a. A **routine** usually represents part of a trial sequence (e.g., fixation cross). **multiple routines** together represent a **trial sequence**.
   b. We can **loop** over routines to repeat trials. We can also change parts of the trial (e.g., image, condition) for each repetition in the loop. In PsychoPy, we have the option to provide the loop with a file that specifies variable settings for each trial. You will learn more about this in Part 2 of this tutorial.
2. Routine:
   a. This part of the builder shows us which **components** are built into the current selected **routine** and their timeline.

3. **Components:**
   a. **Components** can be chained together inside a **routine**. They represent the core elements of our trial sequence (e.g., text, images, sound, video, etc.)
   b. Each component comes with a unique set of settings that usually include things like timing, content, appearance, formatting, data logging, etc.
4. **Menu:** Allows you to open, save, run experiments and various options that have to do with online experiments.

About The Experiment

In this tutorial, you will create a visual-search experiment using real-world photographs. Participants (you) will search for a specified target in a photograph of a scene. The targets will either be placed in a logical – or **consistent** – location in the scene or placed in an **inconsistent** location (see Figure 1). For example, if you were looking for the alarm clock in a bedroom, a consistent location would be next to the bed on the nightstand, whereas an inconsistent location would be on your pillow. Usually we would track eye-movements throughout this experiment such that we can have a full understanding of what is driving behavior. However, because of the current circumstances, we will have participants perform the search with their mouse.

The details (see also Figure 1): On each trial, a small **fixation-cross** will appear at the top of the screen and participants will have to "look" at this cross and then press the spacebar to start the experiment. The name of the **target** will replace the fixation-cross and then, after a short pause, a **scene** will appear. Participants should try to find the target object within the scene as fast and accurately as possible. Once they find the target, they should **click** on it with their mouse. After each trial, participants receive **feedback** on the actual location of the target before the next trial begins.

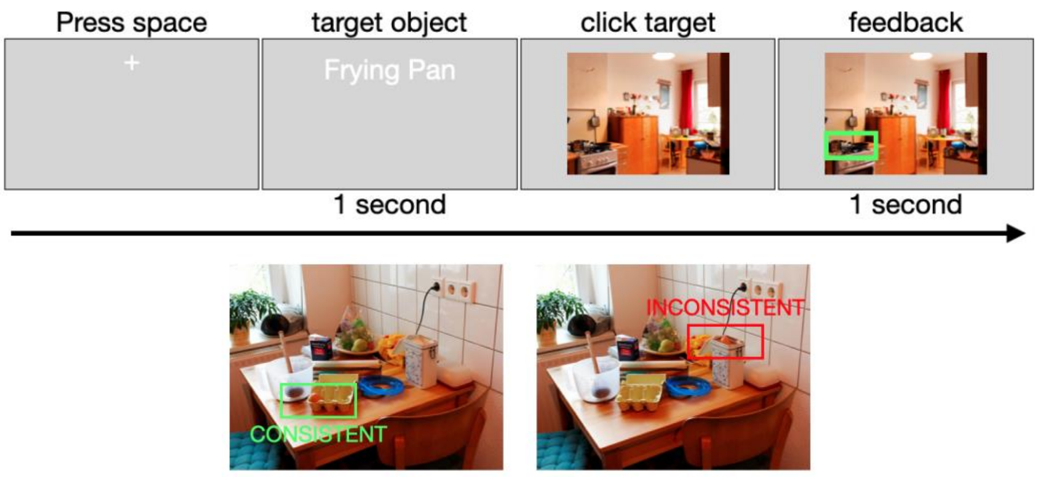


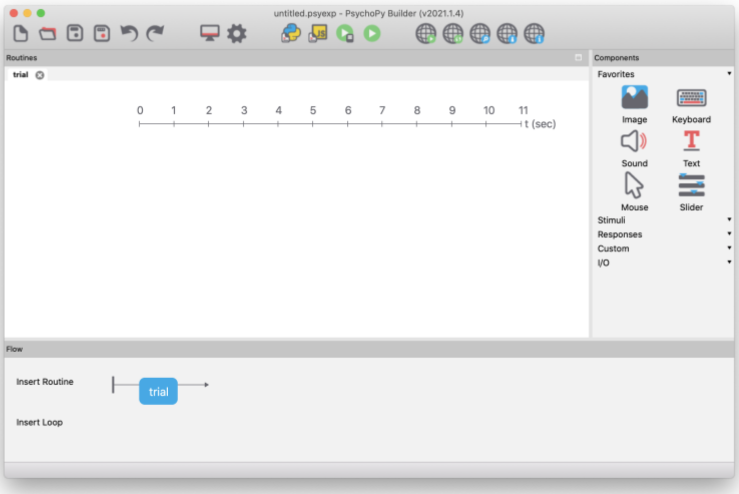
Figure 1: Trial sequence (top) and conditions (bottom) for the visual search experiment.

This tutorial consists of two main parts. In part one, you will build a single trial sequence for the experiment described above. In part two, we will repeat the trial 10 times, with 5 trials per condition (five images in the "consistent" and five images in the "inconsistent" condition).

PART 1 – BUILDING A SINGLE TRIAL

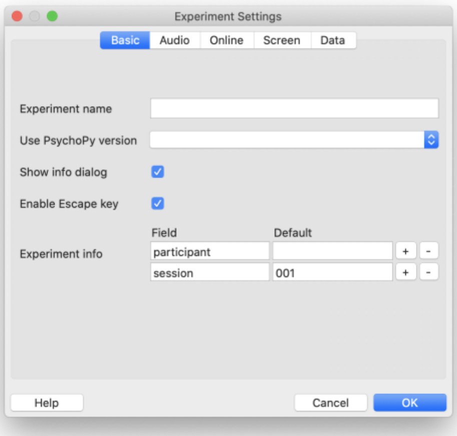
To create a new experiment, open PsychoPy and make sure you're in the **builder view** (at the top of your window it should say something like "untitled.psyexp - PsychoPy Builder", see image below).

Note: the images in this tutorial might look different to what you're seeing depending on your operating system and PsychoPy version.

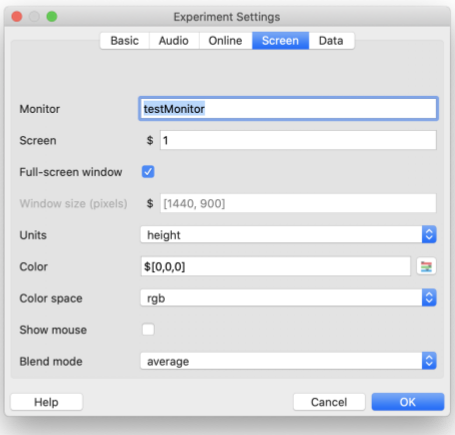


STEP 0 - EXPERIMENT SETTINGS

Before we really get started, let's have a look at the experiment settings:



Experiment name		
Use PsychoPy version		
Show info dialog	✓	
Enable Escape key	✓	
Experiment info	Field	Default
	participant	
	session	001



Monitor	testMonitor
Screen	$ 1
Full-screen window	✓
Window size (pixels)	$ [1440, 900]
Units	height
Color	$[0,0,0]
Color space	rgb
Show mouse	☐
Blend mode	average

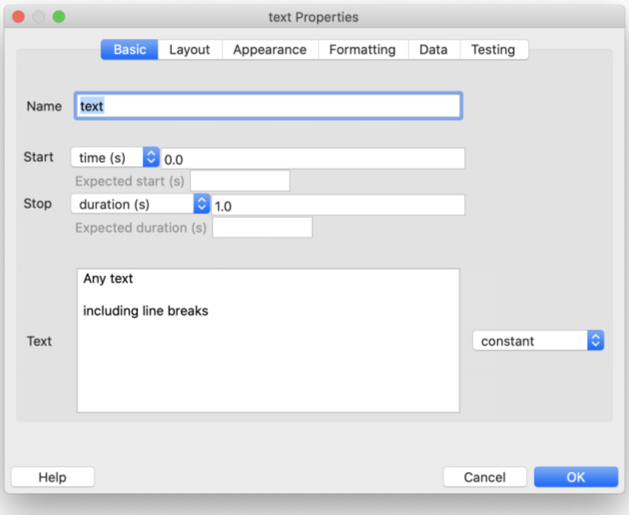
- The first thing you should do is give your experiment a name. For the sake of this tutorial, it does not really matter what name you give it, but something like `visual_search_experiment` or `psychopy_tutorial` would make sense… (Feel free to add some spice though). If you're interested in why we use underscores when naming things and NOT spaces check out this helpful guide: https://slides.djnavarro.net/project-structure/#1.

- Under **Experiment info** you can put in any number you want for the **Default participant** value (that way you don't have to type it in every time you press run). You can also delete the **session** field by pressing the "-" next to it. We don't need it for this experiment.
- Lastly, under the **Screen** tab, make sure that **Units** are set to `height` (should be the case per default). Psychopy offers a range of units to pick from: https://www.psychopy.org/general/units.html. We use height units here because they scale with window height so things will look good even if your window has different dimensions to mine! Don't forget to click **OK** once you're done to save the new settings!
- Now go ahead and **save** your freshly created builder experiment into your "psychopy_vis_search_experiment" folder.

In the next few steps, we will follow the experiment's trial sequence and create a **routine** for each part of the trial. That means we should start by building in the **fixation-cross** that will be presented at the beginning of each trial.

STEP 1 – FIXATION-CROSS

PsychoPy already created a routine called `Trial` for us when we started it. Can you find it? We can just right click that routine and rename it to `present_fixation_cross`. Now, let's fill up this routine with all the components we need to present a fixation-cross. A fixation-cross is basically just the letter X or + (whichever you prefer…). In PsychoPy, this means we need a simple **Text component** to represent our fixation cross. On the right side, under **Components,** select the **Text component** (as part of the "stimuli" components), which will open a window:



Name: text

Start: time (s) 0.0
Expected start (s)

Stop: duration (s) 1.0
Expected duration (s)

Text:
Any text

including line breaks

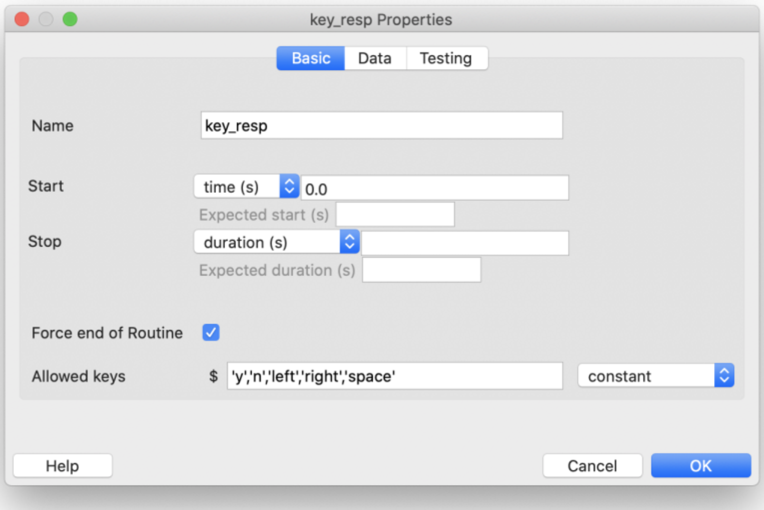
constant

There are a few things we need to change here:

- You can give the component a meaningful name, something like `fixation_cross`
- Delete the `1.0` next to **duration**. We don't want the fixation-cross to disappear after 1 second, instead we want the participant to press the space key to start the trial.
- In the text box simply put `+` or `X`.
- Under the **Layout** tab, set **position** to `(0,0.4)` – make sure you're including the brackets! Some info on coordinates in Psychopy: when working with height units for a standard widescreen with aspect ratio 16:10 the bottom left corner of your screen will be at `(-0.8,-0.5)`, the top right corner at `(+0.8,+0.5)` with the center at `(0,0)`. The first number in the brackets refers to the X-coordinate (left-right) and the second number refers to the Y-coordinate (up-down). So, our fixation-cross with `(0,0.4)` will appear at the top of the screen above the center.
- Under the **Formatting** tab, set **Letter height** to `0.05`

Remember, we want the participant to look at the fixation-cross and press the **spacebar** to begin the trial. In PsychoPy, we can use a Keyboard component to accomplish exactly that.

Select keyboard component from the **Responses** components:



key_resp Properties

Basic | Data | Testing

Name: key_resp

Start: time (s) 0.0

Expected start (s)

Stop: duration (s)

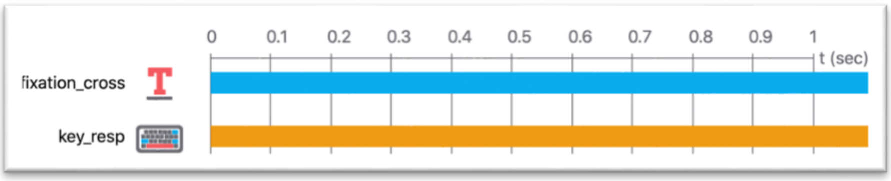
Expected duration (s)

Force end of Routine: ✓

Allowed keys: $ 'y','n','left','right','space' constant

All you must do here is change the **Allowed keys** to only include `'space'`.

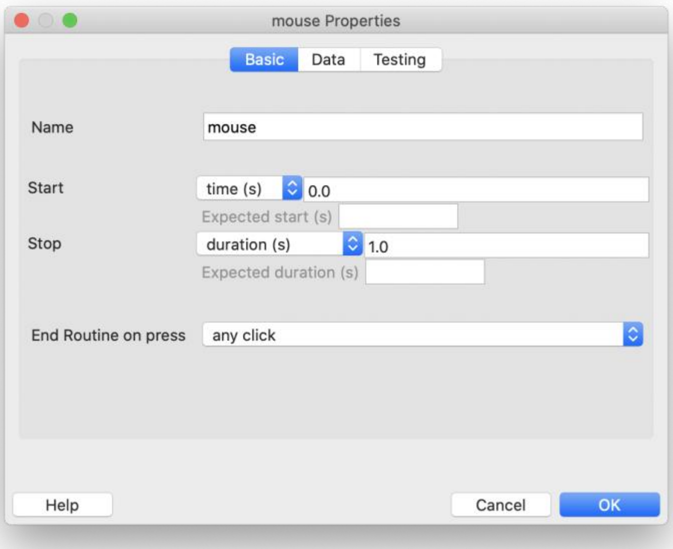
And that's it for the first **routine**! You have successfully implemented the first part of the trial sequence! Your `present_fixation_cross` routine should now look like this:



You can test it out by pressing the green play button. First, the dialogue field will pop up asking you to put in a participant number (with the default number displayed). Once you press **ok** the experiment should start. What you should see then is a gray screen with a white fixation cross on the top of the screen that will stay there until you press the spacebar. After pressing the spacebar, the experiment should close.

Usually, in a lab-based eye-tracking experiment when presenting the fixation cross, we would build in a check that would only allow the trial to continue if the participant is actually looking at the fixation-cross while pressing the spacebar. In our case, we don't have an eye-tracker and therefore no way to control where people are looking at. Instead, participants are searching with the **mouse** by clicking on the target object. So, the least we can do is make sure that at the beginning of the trial, the mouse position is automatically reset to the position of the fixation cross, mimicking what we would do in an eye-tracking experiment.

In your `present_fixation_cross` routine, insert a **Mouse component** (you can find it under **Responses**):



mouse Properties

Basic | Data | Testing

Name: mouse

Start: time (s) 0.0

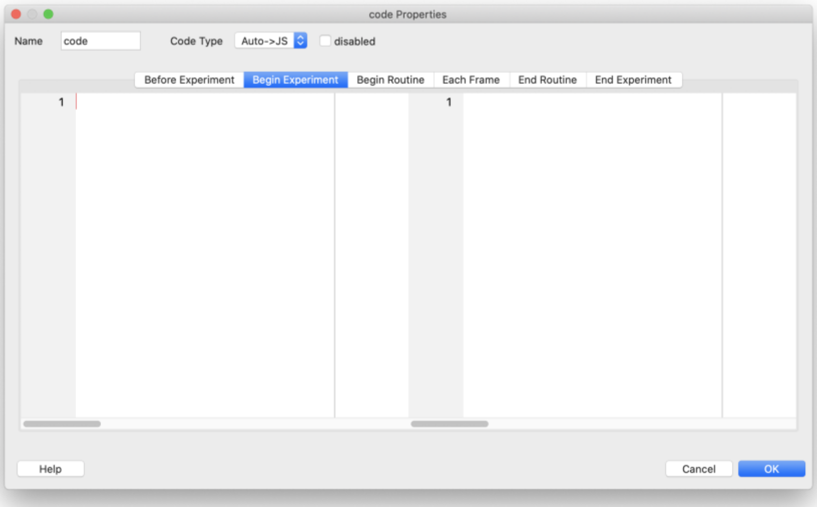
Expected start (s)

Stop: duration (s) 1.0

Expected duration (s)

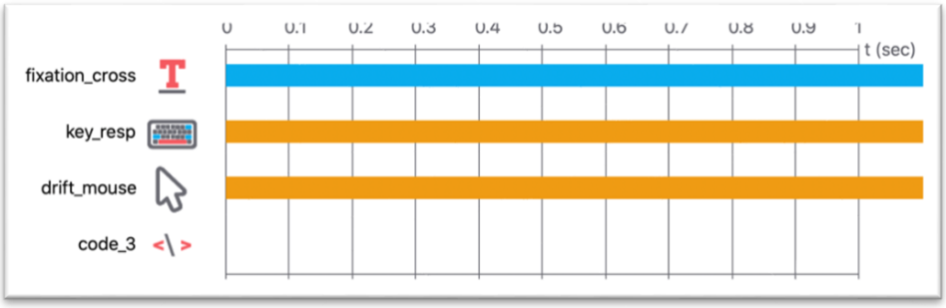
End Routine on press: any click

- Name it `drift_mouse` and delete the `1.0` duration.
- **End Routine on press** select `never` and press **OK**. If we don't do this and the participant accidentally clicks the mouse it would end the routine. However, we have already implemented a keyboard component to allow the participant to continue to the next routine, so we don't want the mouse to do the same thing.
- We will now use some **python code** to control the mouse position. Remember, we want the mouse to automatically be reset to the fixation cross at the beginning of each trial. To do this, select a **Code component**. You can find it under **Custom** components.



- For **Code Type** select `Py` (the other options are important for when you're doing online experiments but it can be confusing so we will just stick to plain Python)
- then go into the **Begin Routine** tab and in the empty text field type:
  `drift_mouse.setPos((0,0.38))`
  This will make sure that in the beginning of the fixation cross routine, the mouse position will be set to slightly below the fixation cross (which is at position `(0,0.4)`). We put it slightly below so that the target object that replaces the fixation cross is still readable.

Your updated `present_fixation_cross` routine should now look like this:



Now give it a go and see if it worked!

STEP 2 – SEARCH TARGET

Next up in the trial sequence (Figure 1) we want to present the search target, which we will build in as a new routine. To include a new routine, just click **Insert Routine -> (new)**, name it `present_target` and put it after `present_fixation_cross`. See how you can switch between the routines by clicking the blue boxes? Clicking on them will open the components included in the selected routine.
We want the search target to replace the fixation cross. For this, we need another **Text component**

- In your `present_target` routine, insert a new **Text component** and name it `target_obj_name`

- You can leave the duration the way it is (should be 1 second per default)
- As for the actual content of the textbox write down `egg` for now.
- Set the **letter height** to `0.05` and **position** to `(0, 0.4)`. Check above if you don't remember how to do that.

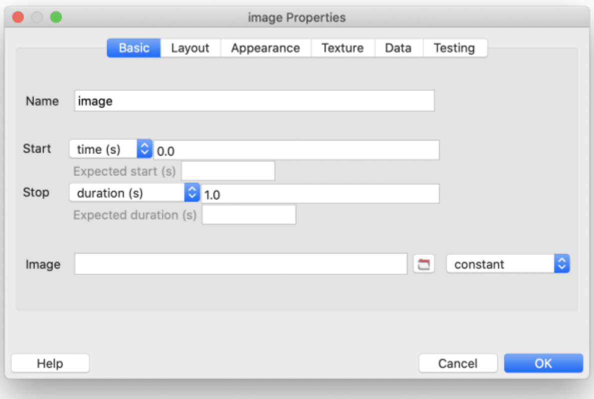
And that's it for the second part of the trial sequence!

Run the experiment to check out if everything looks as you would expect it to. Don't forget to save your progress!

STEP 3 – PRESENT SCENE, FIND TARGET!

Finally, let's get into the actual search part of the experiment! Before reading further, try to think of all the parts of the visual search (everything up to the feedback) that we need to implement and which components we could use for that, maybe even take some notes. This will make it easier for you to understand the next part.
Ok, now that you have a good idea of what parts we need to take care of, let's get started!

- Remember, we're in a new part of the trial sequence, so what do you need to do first? That's right, insert a new **routine** and call it `present_search_image`, put it after `present_target`. Click on it, so you can edit components in the routine.
- To present an image, we can use the **Image component**



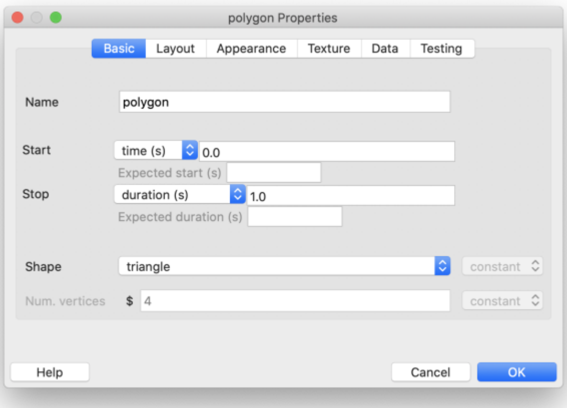
Name: image
Start: time (s) 0.0
Expected start (s)
Stop: duration (s) 1.0
Expected duration (s)
Image: constant

- Give it a useful name, something like `search_image`.
- Delete the duration, we want the image to stay on screen until the participant clicks somewhere (we will take care of that soon).
- Next to **Image**, you need to specify the *path* to the image in your "images" folder. The image we will chose for this trial is named "`C_20CON.png`" which contains the egg as target object.
  - You can either press the folder button next to the empty field to select the image by hand or just type in `images/C_20CON.png`
- Finally, go to the **Layout** tab and set **Size** to `(1.2,0.9)`.

We need to somehow register whether the participant clicked on the target or somewhere else. The most common way to do this in eye-tracking experiments is by using so-called **area**

of interests (AOIs). These AOIs are basically just coordinate boxes around relevant areas (usually objects) in the image. In our case, we only have one relevant object: the target. We will build in an invisible box (polygon) around the target object representing its AOI. We will make it invisible so that the participant doesn't see it, but we can still use it to register where they clicked.

Squares, rectangles, and other polygons can be added with a Polygon component that you can find in the Stimuli components:



polygon Properties

Basic | Layout | Appearance | Texture | Data | Testing

Name: polygon

Start: time (s) 0.0
Expected start (s)

Stop: duration (s) 1.0
Expected duration (s)

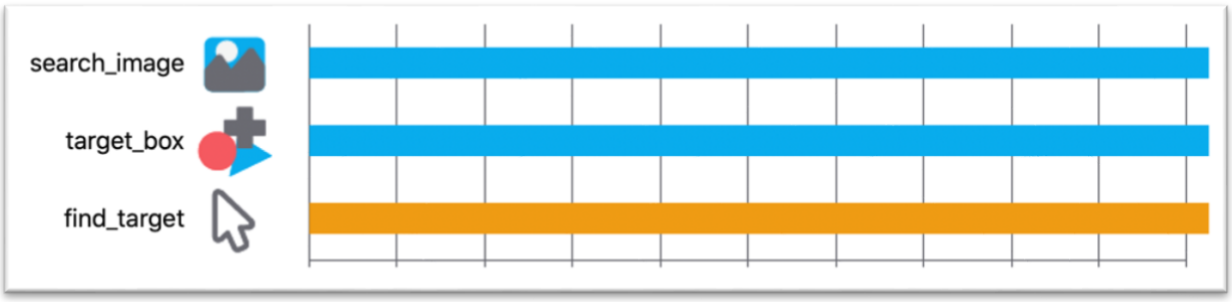
Shape: triangle — constant

Num. vertices: $ 4 — constant

Help | Cancel | OK

- Name it `target_box` and delete the duration.
- Change **Shape** to `rectangle`.
- In **Layout** change **Size** to `(0.08,0.08)` and position to `(-0.22,-0.15)`.
- In **Appearance**, delete **Fill and Border Color**. That way, we can make the rectangle invisible to the participant.
- To register where the participant clicked, we will use another Mouse component.
  a. Change **Name** to `find_target` and remove the duration.
  b. **End Routine on press** should be set to `any click`. That way, the trial will end as soon as the participant clicks on something. If we select `valid click` the trial would only end if they clicked a **clickable stimulus** (see c.)
  c. Next to **Clickable stimuli** you should write `target_box,` (NOTE: the comma here is important!!) Any stimulus component we include here will be registered together with the mouse click and can be accessed later on (we will make use of this during the feedback routine!)
  d. Under the **Data** tab, **Save mouse state** should be set to `on click`.
  e. **Time relative to** should be set to `Routine` (that way we will get a reaction time).
  f. **Save onset/offset times** should be checked.

If you give it a go now, you will not notice the target box, but what should happen is that the image stays on the screen until you click somewhere. We will make use of our invisible box next when creating the **Feedback Routine**.
This is what your `present_search_image` routine should look like right now:



STEP 4 – FEEDBACK

Like before, think about the steps that are necessary for us to present feedback to the participant: we need to know whether they clicked on the target or not and then show them the target with some sort of notification of whether they were correct or incorrect.

The good thing is, we already took care of most of these things in the previous routine! All we need to do now is:

- Create a new routine, put it behind `present_search_image` and call it `present_feedback`
- PsychoPy allows you to copy-paste components within and between routines! Go back into the `present_search_image` routine and right click the `search_image` component **-> copy**.
- Now go back to the `present_feedback` routine, at the top of the builder go to **Experiment -> Paste Component**. It will ask you to give the components a new name, chose something like `search_image_feedback`.
- In your new `search_image_feedback` component set **duration** to 1.
- Do the same for the `target_box` component (chose something like `target_box_feedback` as the new name).
- In your new `target_box_feedback` component, set **duration** to 1.

Now, we just need to put in some code to determine whether the correct object was clicked and change the border color of the `target_box_feedback` to green or red respectively:

- In your `present_feedback` routine, add a **code component** and set **code type to** Py.
- Right click the **code component** and select `move to top`
- In the **Begin Routine** tab put in the following lines of code:

```
if find_target.clicked_name:
    box_color = "green"
    accuracy = 1
else:
    box_color = "red"
    accuracy = 0
thisExp.addData('accuracy', accuracy)
```

Let’s try to figure out what is happening in the code above

Aylin Kallmayer – September 2021

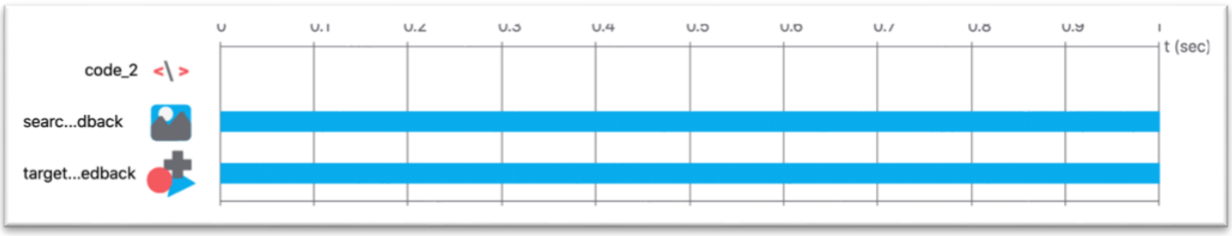
- find_target refers to the Mouse component we added in the present_search_image routine.
- .clicked_name returns a list of clicked stimuli. Remember how we included target_box as a clickable stimulus for the mouse component? If the mouse was clicked inside the target_box then target_box will be returned by find_target.clicked_name. If not, the list will be empty.
- By that logic, the first line "if find_target.clicked_name:" checks whether the list of clicked stimuli is empty or not. If it is not empty (statement evaluates to True, the mouse was clicked inside target box) accuracy evaluates to 1 and box_color is assigned "green". Otherwise, accuracy evaluates to 0 and box_color is set to "red".
- Both box_color and accuracy are variables we create here inside the experiment that we will have access to from now on.
- thisExp.addData() lets you add variables you create inside the experiment (like accuracy) to the output data file (they won't be added automatically so we need to include this step here).

I know, this was probably a lot to wrap your head around, but there is one more thing we need to do, which is assigning the target_box_feedback border color to correspond to the variable box_color (created in our code).

- In your target_box_feedback component, go to the **Appearance** tab and set **Border color** to $box_color. The dollar sign $ signals to PsychoPy that we don't want to use an absolute value, but rather a **variable** whose value depends on the participant's behavior.
- To be able to flexibly change **Border Color** every trial, according to the value of box_color, we need to change it to set every repeat instead of the default constant.
- You should also change the **Line width** to something like 6 (you can play around with the line width later if you want to make it thicker or thinner).

You have implemented every part of the trial sequence for the visual search experiment! Give it a go and see if everything looks as you expected!

Sanity check: this is what your present_feedback routine should look like:

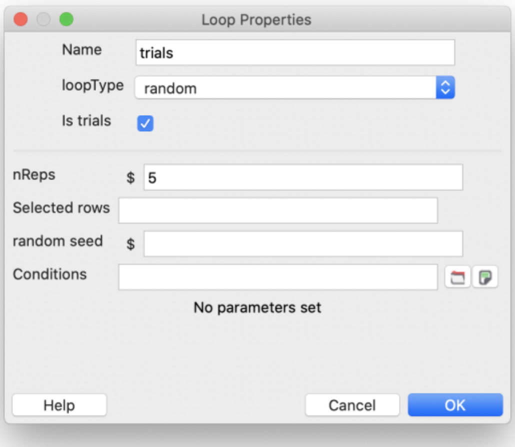


PART 2 – LOOPING

Let's recap what we learnt at the beginning of this tutorial, because it's been a while: we have one independent variable `condition` with two levels: `consistent` vs. `inconsistent` (see Figure 1).
In other words, for each scene, there are two photographs, one where the target is in a consistent location and one where the target is in an inconsistent location (e.g., `C_4CON.png` and `C_4INCON.png`). Each participant will complete all the conditions, so this experiment is referred to as **within participant**.
Although we have two versions of each scene, a single participant should only search through one version of each scene – i.e., a participant should not see the same bathroom twice. That means we must **counterbalance** the stimuli across participants (we will take care of this in a second).
So far, we have built a single trial, so the first thing we need to do is insert a **loop** that repeats that single trial multiple times for each scene.

STEP 1 – INSERT THE LOOP

In PsychoPy, click **Insert Loop** at the bottom of the builder and put the start point in front of `present_fixation_cross` and the end point after `present_feedback`.


Loop Properties

Name	trials
loopType	random
Is trials	✓
nReps	$ 5
Selected rows	
random seed	$
Conditions	

No parameters set

- Press the folder button next to the Conditions field. This lets you specify the condition file:
  `trial_loop_table_200721.xlsx`
  It's worth having a look at the file before you continue. This file contains all the information for each trial (each row represents a single trial) and after loading in the file into the loop, we have access to all the variables contained in the file.
- Next, replace the `5` next to **nReps** with `1`. That means, PsychoPy will go through the file once (in random order: see loopType).

STEP 2 – VARY TARGET OBJECT

Now we can use the variables from the file that we just loaded into our loop to change the target object and scene for each repetition. If we don't do this, we would still just see the egg trial over and over again for 10 repetitions. Loading in the file just gives us **access** to the variables, but we still have to take care of actually implementing them in the appropriate components ourselves. Luckily, this step is pretty straight forward in Psychopy. Let's begin with varying the target object:

- Go to `present_target` routine
- Click on the text component `target_obj_name`
- Replace `egg` with `$target` – if you remember the `$` signifies a variable, in this case the variable comes from our condition file.
- If we want the target to change every trial, we need to also change `constant` to `set every repeat`

Go ahead and try running the experiment now! The target object should now change for every repetition, everything else should stay the same.

STEP 3 – VARY SCENE

Now it will get a little trickier – we're getting into the actual **counterbalancing**. Let's reiterate: there are 20 images (10 different scenes with consistent and inconsistent target locations each), but each participant will only get to see each scene once meaning 10 images overall. How do we decide which participant gets to see which image? The solution is **counterbalancing**. If participant 1 saw image 1 in the consistent condition, participant 2 should see image 1 in the inconsistent condition and so on. We already took care of this in the condition file where you can see we have two conditions with alternating consistency. Now, all we must do is assign half the participants to `condition_1` and half to `condition_2`. An easy way to do this is by dividing participants into even and odd participant numbers.

- Go into `present_search_image` routine
- Insert a new **Code Component** <\>
- Right click the **Code component** and select `move to top`
- Set **Code Type** to `Py` and go into the **Begin Routine** tab
- Put in the following lines of code:

```python
if int(expInfo['participant']) % 2 == 0:
    condition = condition_1
elif int(expInfo['participant']) % 2 == 1:
    condition = condition_2
path = "images/"+stimulus+condition+".png"
if condition == "CON":
    position=position_con
    size=size_con
elif condition == "INCON":
    position=position_incon
    size=size_incon
thisExp.addData('condition', condition)
```

Let’s go through this piece of code step by step again:

- `%` in python represents **modulo**. Any number modulo 2 will evaluate to 0 if the number is even and to `1` if the number is odd. So, `2 % 2` evaluates to `0` and `3 % 2` evaluates to `1`. So, the first expression basically reads as “if the participant number is even then choose whatever `condition_1` is for this trial (can be either `“CON”` or `“INCON”`) and assign it to the new variable `condition`.
- In python, strings (characters) can be “added” together with `+`. We can use this to our advantage and define a `path` variable which consists of 4 parts added together: `“images/” + stimulus + condition + ”.png”`
    - So, for example if `stimulus` (from the condition file) is `C92` and `condition` for this trial is `“CON”` then `path` will evaluate to `“images/C92CON.png”` Therefore, we can now use `path` to define the image in our image component for each trial.
- The second if-statement selects the appropriate `position` and `size` variable for the AOI depending on `condition`.
- The last line takes care of logging the condition for each trial in our data file.

Now, we can use our newly created variables `path`, `size`, and `position` in the respective image and polygon components:

- Edit the `search_image` component, replace `images/C_20CON.png` with `$path`
- Replace `constant` with `set every repeat`
- Do the same to the image component in the `present_feedback` routine.
- In `present_search_image` routine edit the `target_box` component:
    a. Under **Layout** set **size** to `$size` and **Position** to `$position`
    b. Set both from `constant` to `set every repeat`
- Do the same in the `present_feedback` routine.

And you’re done! Give it a go

FINAL TOUCHES – INSTRUCTIONS

Your experiment should include some instructions in the beginning that explain how the task works. Add a new routine before your experimental loop that includes a text component with instructions and a keyboard component that allows participants to continue by pressing space. You can also include some “end of experiment” routine that thanks the participant for their participation and lets them know that they have completed the experiment.

RUNNING THE EXPERIMENT

You’re all set up to run the experiment. Remember, the counterbalancing depends on the participant number that is entered in the beginning of the experiment when the dialogue box pops up. Keep track of participant numbers to make sure you have a balanced number of participants per condition.

FURTHER RESSOURCES

This was a very compact introduction into PsychoPy builder experiments. PsychoPy has many builder demos that you can check out. I also recommend checking out this 2-day python + PsychoPy workshop: https://github.com/aylinsgl/GRADE_Python_Workshop_110521
It has some more in-depth information and exercises for programming in python and using the PsychoPy Coder which we haven't looked at yet.

Aylin Kallmayer – September 2021