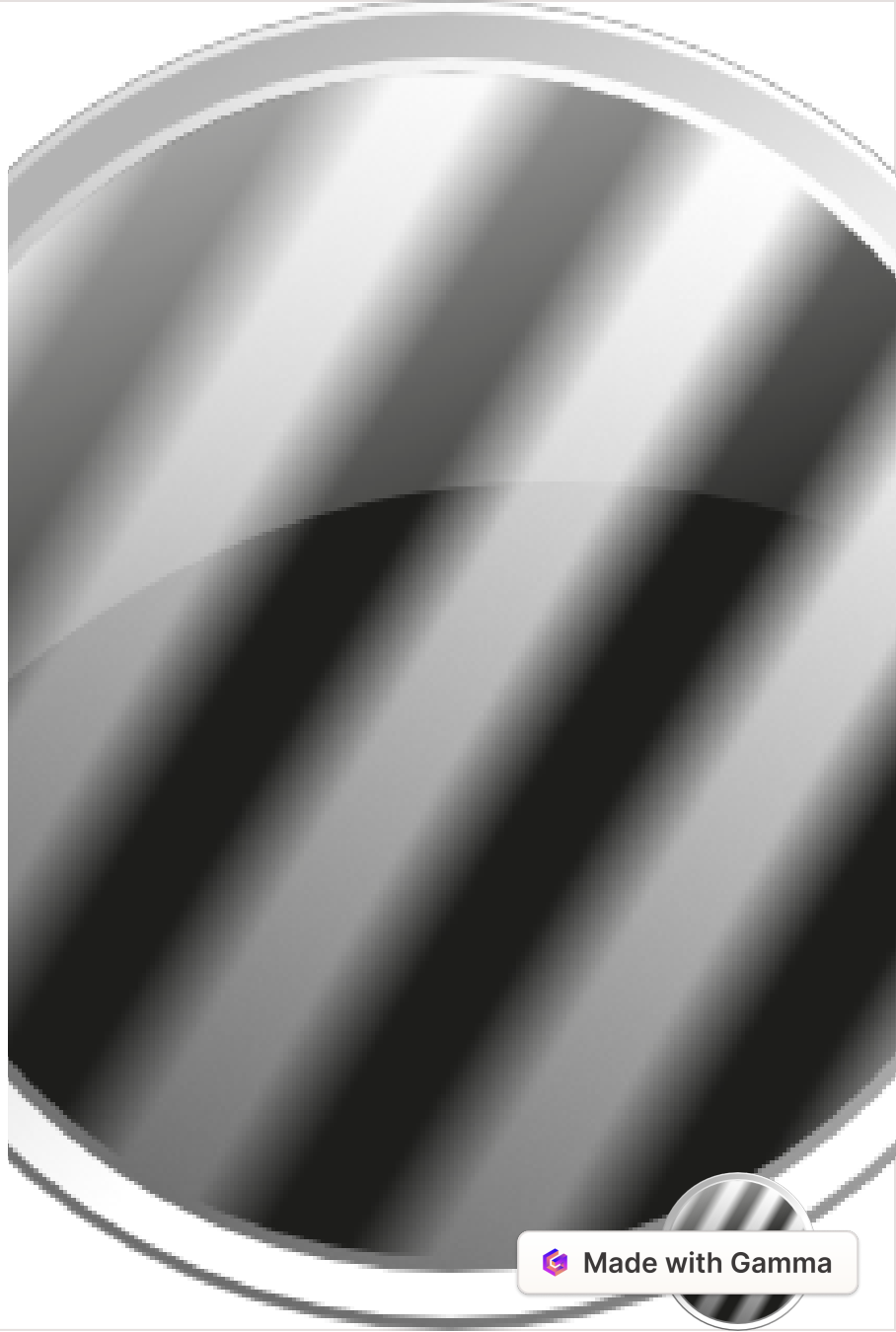
Creating an Experiment in PsychoPy

In this session, we will explore the process of creating an experiment using PsychoPy, a powerful and user-friendly software for behavioural research.

by Becca Hirst



Session content

- Introduce PsychoPy.
- The basics of presenting stimuli and gathering responses (response times and accuracy).
- Updating stimuli trial-by-trial.
- Making stimuli "dynamic" .

Introduction

PsychoPy is an Open Source project, the users have access to the code and can contribute at any time! That kind of means the team is technically huge!

- Github: > 150 contributors
- Funders: University of Nottingham. Wellcome Trust and Chan Zuckerberg Initiative
- Forum: Discourse.psychopy.org
- Staff: Open Science tools staff

What is PsychoPy?

It's *Psychology software in Python.*

PsychoPy is a Python library, an script editor (Coder) an application with a GUI (Builder). It is, itself, entirely written in Python

2002 - 2003

Jon began work on this for his own lab.

2003 - 2017

Purely volunteer-driven, evenings and weekends project

2017 - now

Still open source and free to install but with proffessional support.

What is the goal of PsychoPy?

The aim is to enable scientists to run as wide a range of experiments as possible, as easily as possible, with standard computer hardware.

A single piece of software:

- Precise enough for psychophysics
- intuitive enough for undergraduate psychology
- flexible enough for everything else
- capable of running studies in lab or online

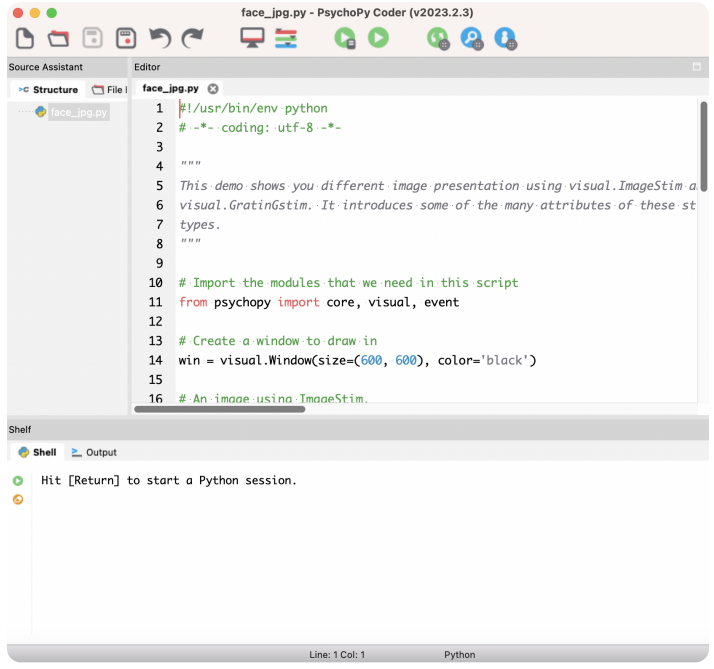
Choice of interface

Builder view

Why do people build?

- It is far faster to develop experiments!
- You can still understand and build on the experiment next year.
- You'll probably have fewer bugs.
- Code components can be added whenever Builder is not enough.
- Your experiment can compile to Python or Javascript.

We (the team) use builder with code components where needed.



Coder view

Why do people code?

- To implement more complex experimental designs/procedures(?)
- To know exactly what the code is doing(?)
- To break out of the “trials/blocks” structure or drawing loop cycle
- To program things that are not experiments (e.g. data analysis)

Additional helpful resources

- Textbook (designed for undergrads to research staff): Building Experiments in PsychoPy
- Forum: discourse.psychopy.org
- YouTube: psychopy_official

Let's start making experiments!

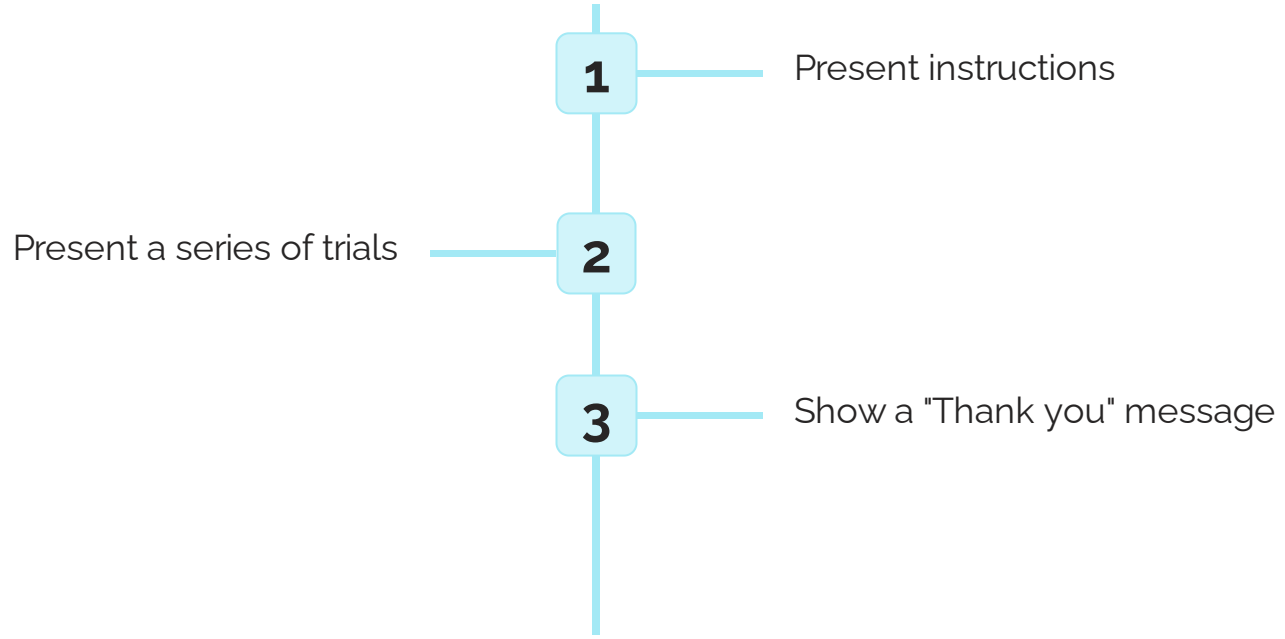
What you will need:

- A **recent** version of PsychoPy installed
- A pavlovia account (use your institutional email!)

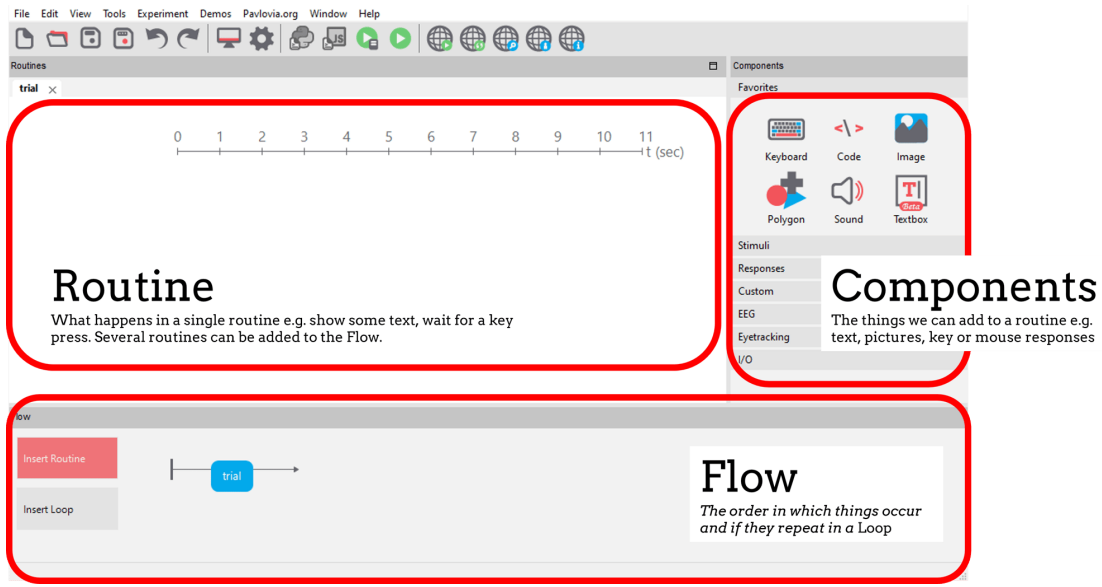
> If you are on Mac, then you might need to give PsychoPy permission to access your keyboard. You can do this using System Settings > privacy & Security > Accessibility > ensure PsychoPy is enabled.

Plan the flow of your experiment

Before starting to create an experiment, we should consider the "flow" of our experiment



What makes a PsychoPy experiment?



- Add Routines to your flow e.g. instructions, trial, thanks (using **insert routine**)
- Add components to your routine e.g. a stimulus, a response (select from the component panel)
- Configure the parameters of each component e.g. stimulus position and size.

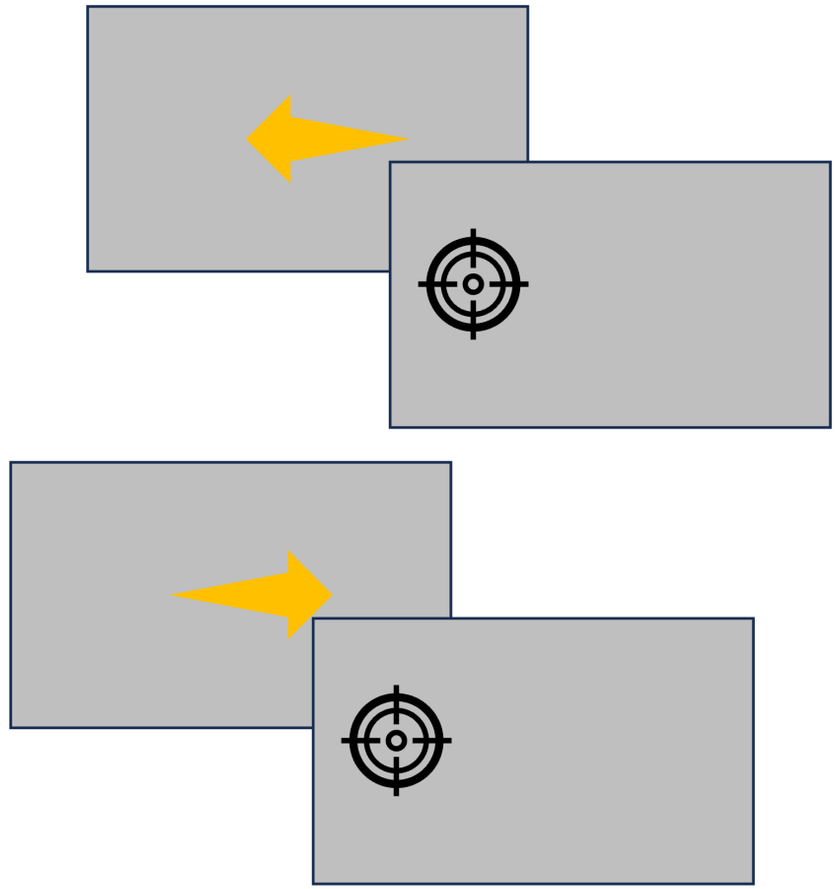
Setting up your trial

Example: A Posner cueing task. An arrow cues the location of a target. The participant must respond using the left and right arrow keys.

What component make up this routine?

- An arrow/cue
- A target
- A keyboard response (left/right keypress - respond to the location of the target)

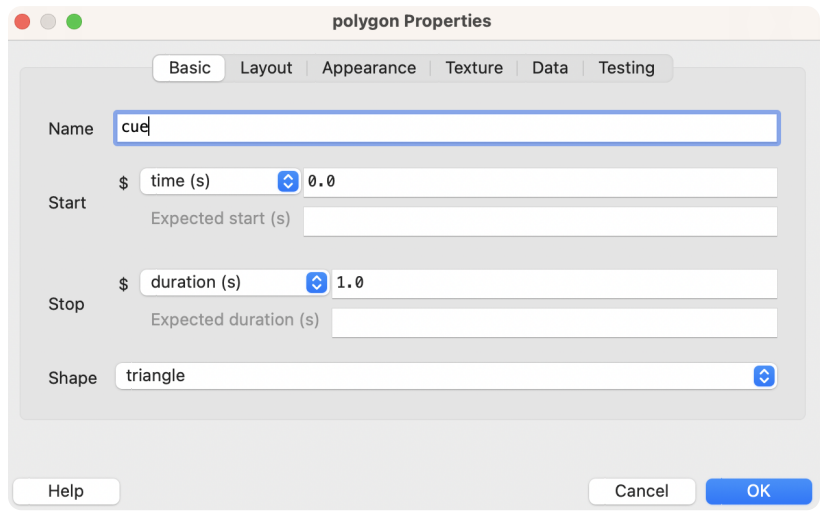
Example trials valid (top) Invalid trial (bottom)



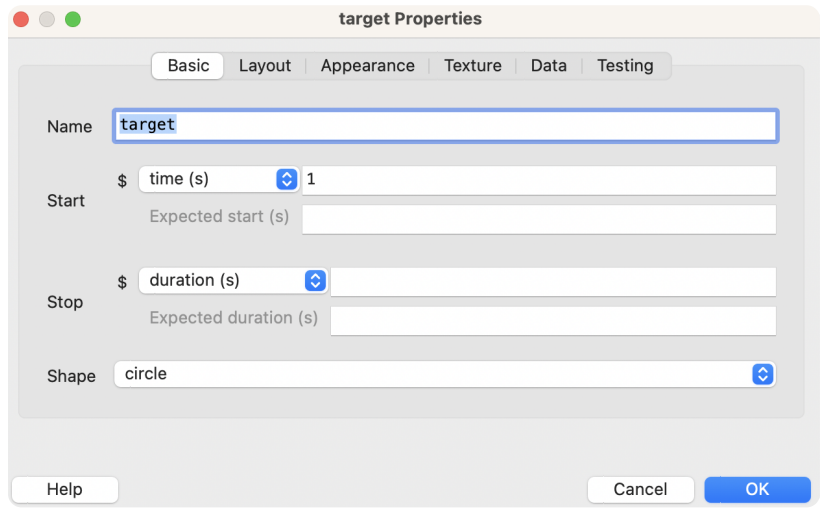
Presenting Shapes

Let's add our cue and target stimuli to the "trial" routine.

Cue stimulus: Add a **polygon** and leave the shape as "triangle". The cue starts at 0 seconds and lasts 1 second. in the **layout** tab set **orientation** to 90.



Target stimulus: Add another **polygon**, set shape to be "circle". The target starts at 1 second and stays onscreen until a response is made. **To make a stimulus stay on screen until a response is made, leave the duration as blank.** In the **layout** tab set the **position** to (0.5, 0)**.**

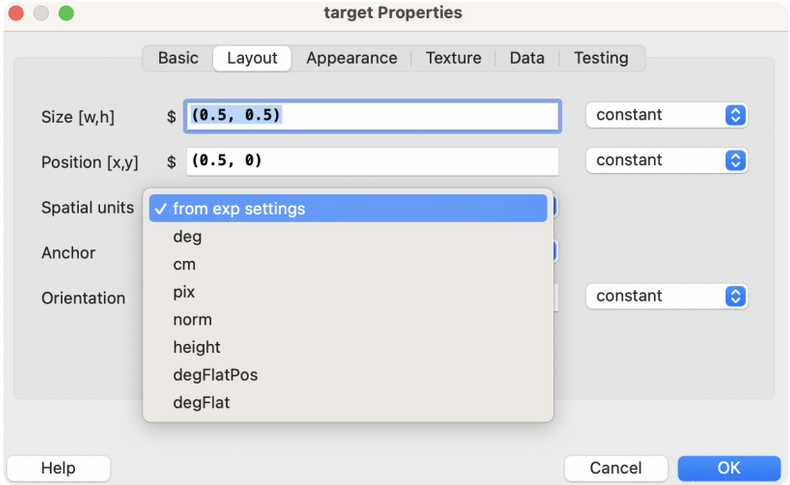


> The x, y coordinate for the centre of your screen is 0, 0. Positive x values are right and negative left. Positive y values are up and negative down. Therefore a position of (0.5, 0) will position your target on the right of the screen.

Understanding PsychoPy units

When controlling the size and position of your stimuli, you want to know what unit stimuli are presented in. You can see what unit your stimuli are using in the **layout** tab. You can see more about PsychoPy units here.

- **from exp settings:** unit inherited from global experiment settings (see next slide). By default we use "height". This means that everything will be scaled proportionally to the window size. a size of 0.5 corresponds to 50% of the window height.



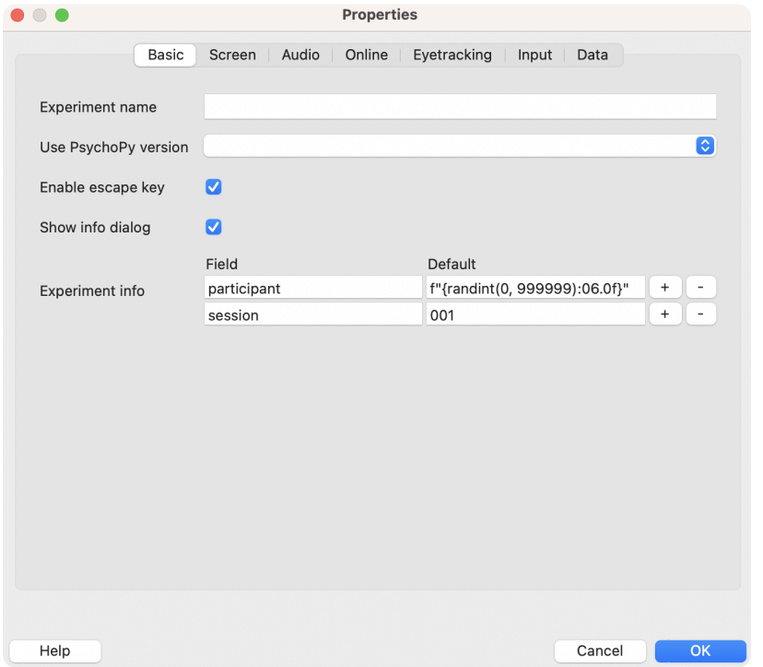
⚠ If you are using any unit other than "height" or "norm" you must configure your monitor settings in the Monitor Centre. For online experiments, we recommend "height" units because this means that stimuli will scale proportionally to the participants screen.

Your Experiment Settings

By selecting the cog icon at the top of the Builder app, you can access the global settings of your experiment.

In the "Basic" tab you can configure your experiment name, what version of PsychoPy you are using and add/remove fields in the "Experiment Info".

In the "Screen" tab you can set up the global units used in your experiment, if you want to use full screen, and which monitor from your **Monitor Centre** you want to use.



Everything in the "Experiment Info" will be presented as options in a dialogue box at the start of your experiment. So you can provide info such as participant ID.

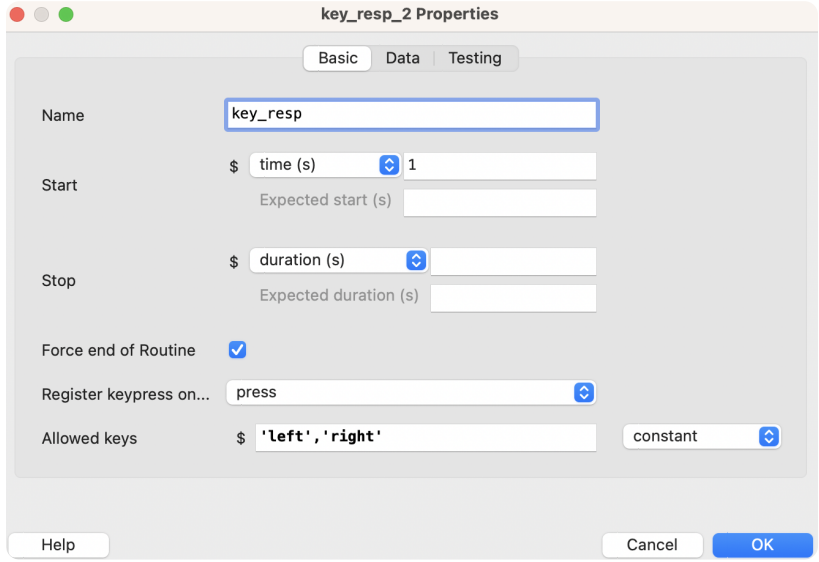
Gather a Keyboard Response

Keyboard responses also have a start time and duration, by default, the duration field is blank (meaning it will last indefinitely).

Force end of Routine: If checked, a keyboard response will end this routine, and the experiment will move on.

Allowed keys: A list of key names to watch (each name must be in quotations and separated by commas). if blank, all keys will be watched.

We will allow the "left" and "right arrow keys to be pressed.



Check it works!

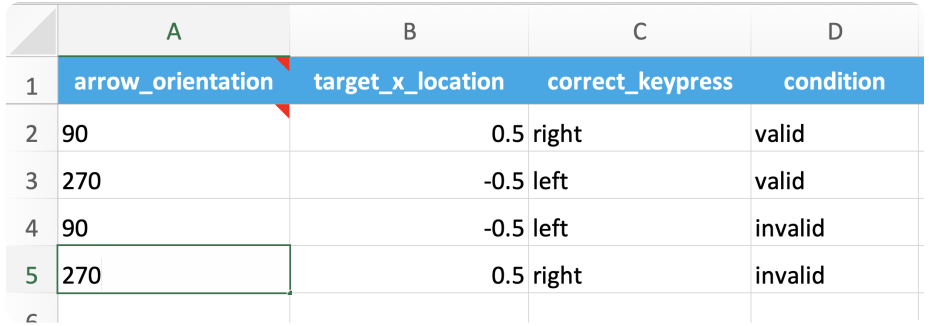
You can run your experiment by pressing the green "run" icon. You should see your cue appear for 1 second, then your target, which should remain onscreen until a response is made.

Setting up your trials spreadsheet

Most experiments consist of a series of trials. Before making an experiment think about what changes trial by trial and make a spreadsheet.

What changes trial-by-trial?

- Arrow orientation (90 or 270 degrees)
- Target location (x coordinate)
- Correct answer (left or right)



	A	B	C	D
1	arrow_orientation	target_x_location	correct_keypress	condition
2	90	0.5	right	valid
3	270	-0.5	left	valid
4	90	-0.5	left	invalid
5	270	0.5	right	invalid

Column headers: Parameters that change trial-by-trial.

Row values: The value of each parameter on each trial.

Repeating a routine e.g. a series of trials

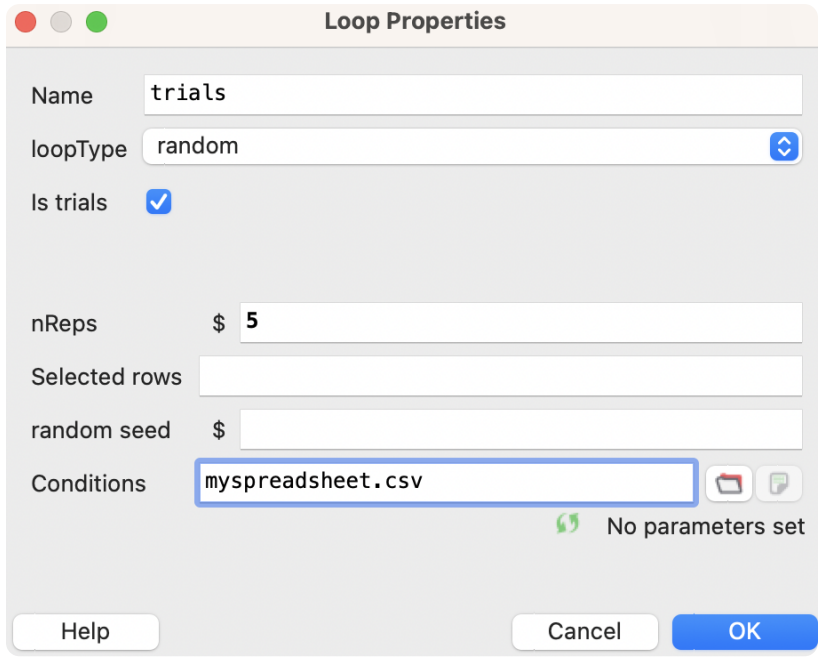
Loop properties

Select "Insert Loop" and select either side of the routine(s) you want to repeat (for us this is the trial routine)

Conditions: A spreadsheet detailing what changes trial-by-trial (see previous slide)

nReps: the number of times the spreadsheet rows are repeated (if no spreadsheet this is just the number of times the loop repeats)

Selected Rows: Row indices from the spreadsheet to present



If you set **nReps to 0** then everything inside that loop will be skipped. This is useful for debugging and branched designs (e.g. if you only want to show part of your experiment)

loopType:

- **random** - randomise rows of spreadsheet - within repetition.
- **full random** - randomise across repetitions of the spreadsheet.
- **staircase/interleaved staircase** - specialised adaptive procedures used typically in psychophysics.

Is Trials: if yes, add new row to data file for every repeat of this loop.

Updating stimuli trial-by-trial

We can now update our stimuli using the variables in our spreadsheet.

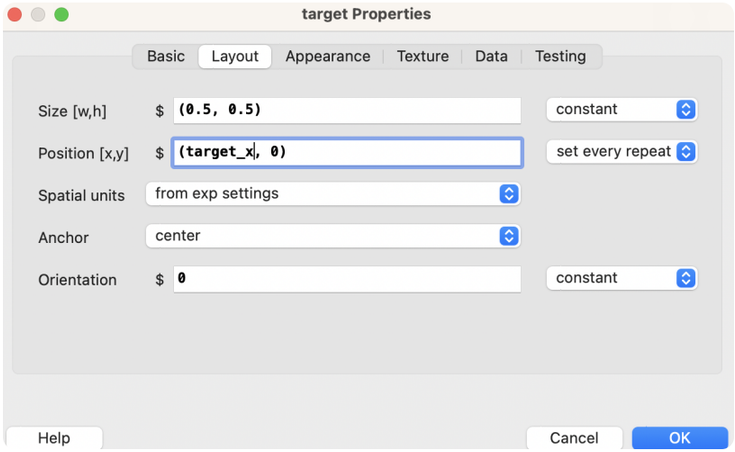
	A	B	C	D
1	arrow_orientation	target_x_location	correct_keypress	condition
2	90	0.5	right	valid
3	270	-0.5	left	valid
4	90	-0.5	left	invalid
5	270	0.5	right	invalid

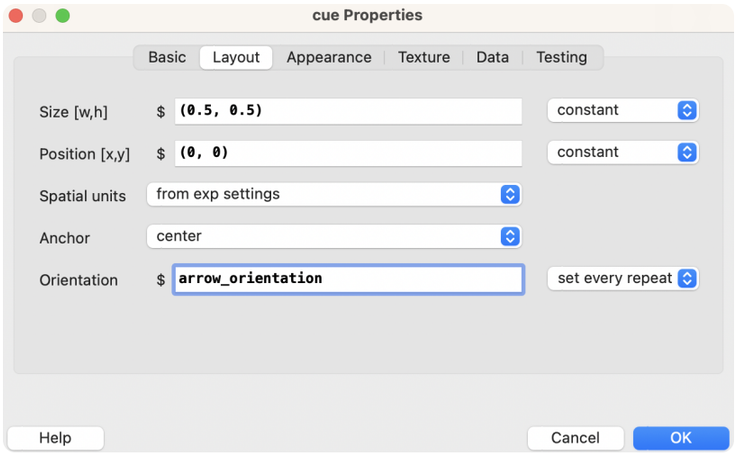
In the **target** stimulus, we can update the x coordinate using **target_x**.

In the **cue** stimulus we update the orientation using **arrow_orientation**.

For any property that is changing trial-by-trial make sure the dropdown at the side is set to **"set every repeat"**

> If you are using a variable to update a component property start the field with a '$' sign only if that field does not already have a '$' in it.s field.





Check it works!

You can run your experiment by pressing the green "run" icon. You should now see the location of your target and the orientation of your cue change trial-by-trial.

Exercise

- Add a new routine for your instructions
- Present some text telling the participant what to do, allow them to press space to start the experiment.
- Add a routine to say "Thanks" at the end of the experiment.
- Add another field to your experiment settings to gather more information (e.g. handedness, age)

Presenting Text

Two components can be used for adding text. **Text** and **TextBox**. TextBox has the added benefit that you can use similar formatting to what you would with a text box e.g. anchor alignment.

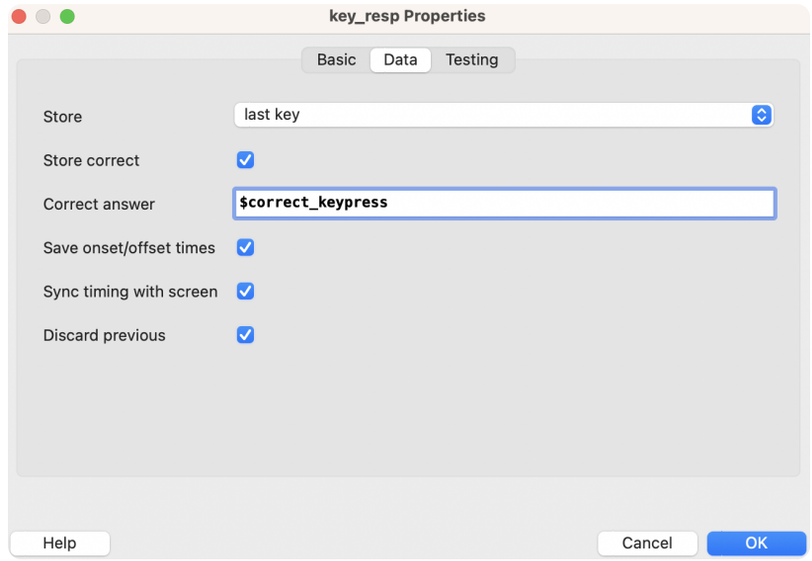
> You can also use the TextBox component to gather typed responses

Storing accuracy from keypress responses

To store whether a participant was correct or incorrect, you can use the column from your trials spreadsheet.

1. In your keyboard component select the "Data" tab.
2. Check "store correct"
3. in the correct answer field feed in the column from your spreadsheet.

	A	B	C	D
1	arrow_orientation	target_x_location	correct_keypress	condition
2	90	0.5	right	valid
3	270	-0.5	left	valid
4	90	-0.5	left	invalid
5	270	0.5	right	invalid



Extending Builder with code

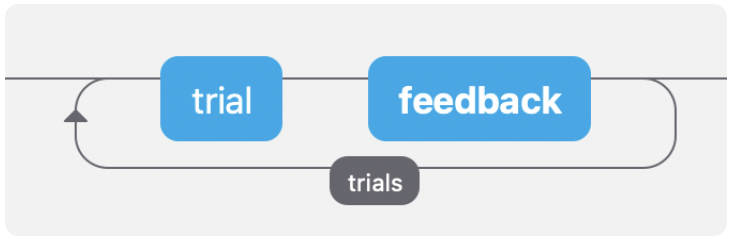
Sometimes you might want to add flexibility to your experiment in ways that might not be accommodated through the GUI interface. This is one of the reasons people may prefer to use code.

In PsychoPy, you can use code components to add flexibility where code is needed.

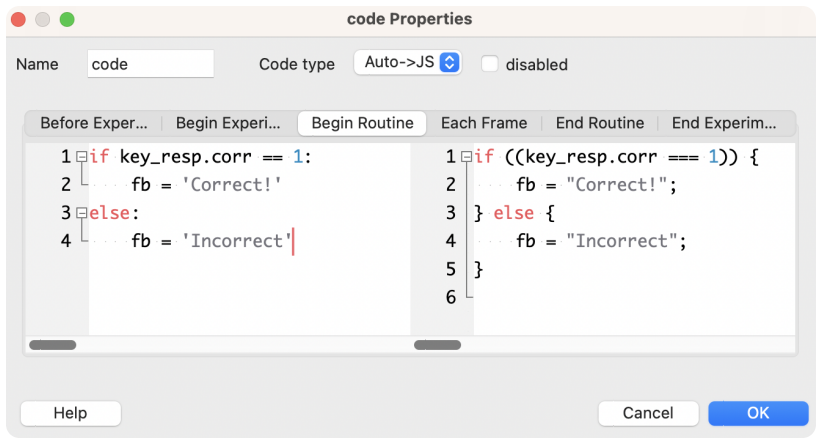
Example: I want to give my participant trial-by-trial feedback (e.g. "**Correct!**" if response was correct and "**Incorrect**" if response was incorrect). In this example we do not know what the text we present will be until a response is made, so we need some conditional IF/ELSE statement to set up our text.

Adding trial-by-trial feedback

Start my adding a new routine to your experiment called "feedback". It should appear after the "trial" routine, but it should occur on every trial, so it therefore needs to be within the loop.



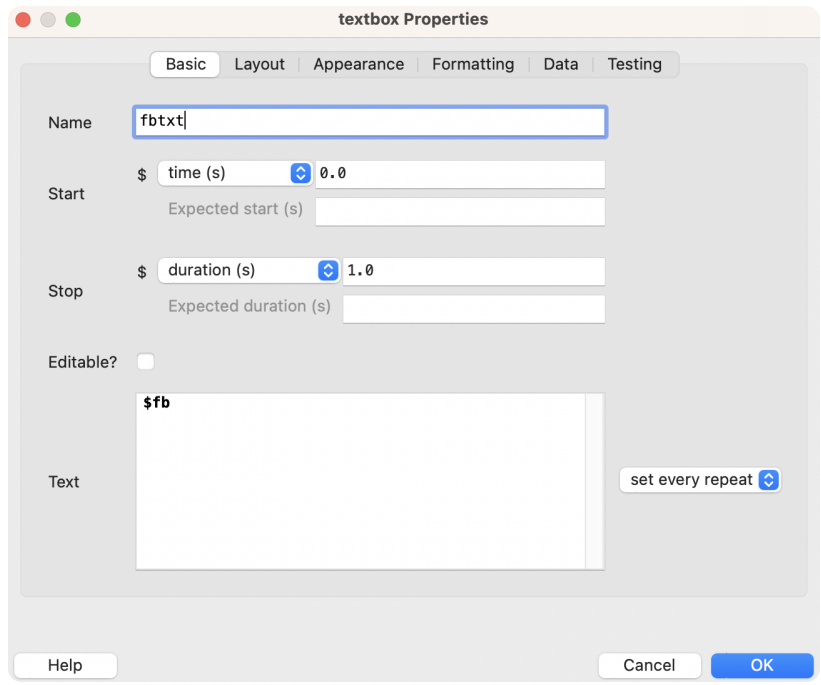
We then add two components to our "feedback" routine. A code component (in the "custom" section of Components) and a TexBox.



Code component

The code component checks if the answer was correct by inspecting your `key_resp` component, which now has the property `corr`. This will be 1 if correct and 0 if incorrect. So we use this to make a variable `fb`.

> ⚠ The tab you add your code to is important. Here we don't know if you are correct until the beginning of your feedback routine. So we add the code there.



Text box component

We use the variable `fb` as the text in our TextBox component, make sure to **set every repeat**.

Exercise

Imagine we want to present images as feedback in addition to our text.

- Find two images (one to use for correct feedback and one to use for incorrect. For ease, you can use the "star" and "asteroid" images in the images folder **here**.
- Save your images to the same location as your experiment.
- Add an image component to your feedback routine.
- Update your code component so that if also sets up a variable called `fbim` - this will be the name of one image if correct and another image if incorrect.
- Make it so your image used `fbim` to update on every trial.

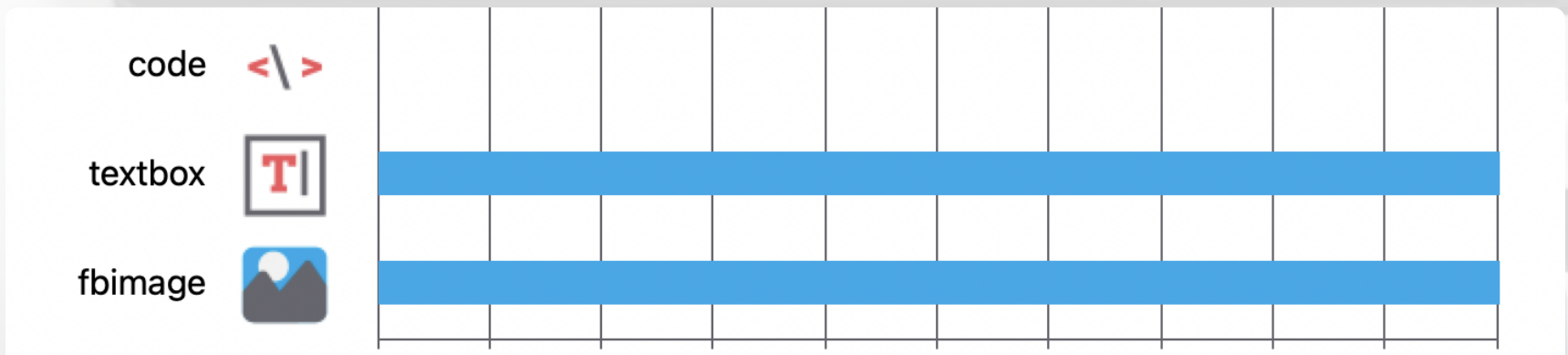
Solution

Your code component should now look like this in the Begin Routine tab:

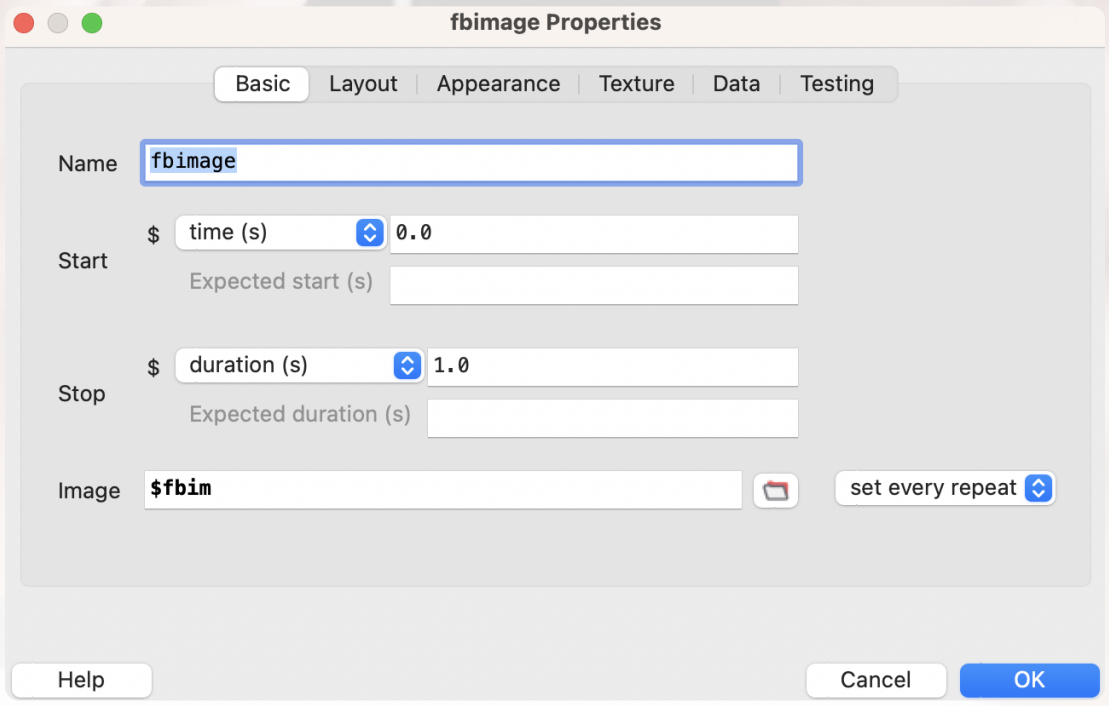
```
if key_resp.corr == 1:
    fb = 'Correct! RT: ' + str(int(key_resp.rt*1000)) + 'ms'
    fbim = 'images/star.png'
else:
    fb = 'Incorrect RT: ' + str(int(key_resp.rt*1000)) + 'ms'
    fbim = 'images/asteroid.png'
```

Here I have a subfolder called "images" where i have an image called "star.png" and an image called "asteroid.png", so I need the full path from my .psyexp file to the image.

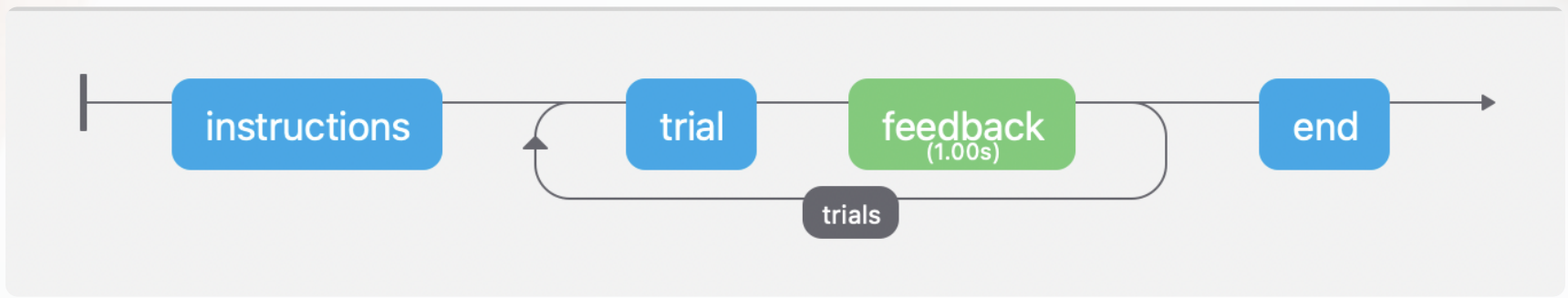
My "feedback" routine looks like this:



And my "fbimage" component looks like this, where it uses the `fbim` variable to update the image trial-by-trial:



My whole experiment flow looks like this:

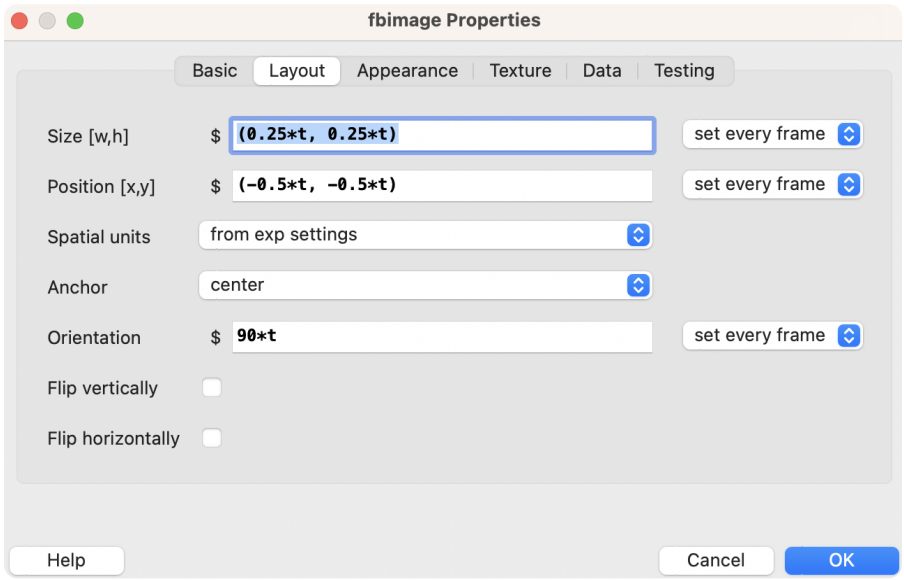


Making "dynamic" stimuli

By making something "Dynamic" we mean making stimuli animated (e.g. move position, change size, spin around). In PsychoPy this can be done easily by using the "set every frame" option in the parameter dropdown of components. This can make for much more interesting and flexible experiments - so let's give that a go!

Accessing current time using "t"

Under the hood of every routine is a timer, the timer starts at 0 at the beginning of your routine and will increment continuously throughout the routine. The timer can be accessed using the variable `t`. Update your fbimage layout with the following:



Spinning stimuli

Update the Orientation to be how many degrees per second you want to spin and set that parameter to "**set each frame**".

orientation: `90*t`

Moving stimuli

Update the x and/or y coordinate to update based on t.

position: `(-0.5*t, -0.5*t)`

Here `-0.5` corresponds to the end x and y coordinate.

Growing stimuli

Update the width and/or height based on t.

size: `(0.25*t, 0.25*t`

Here `0.25` corresponds to the end size of the stimulus.

That's all for this session!

Up next - complex designs and counterbalancing.