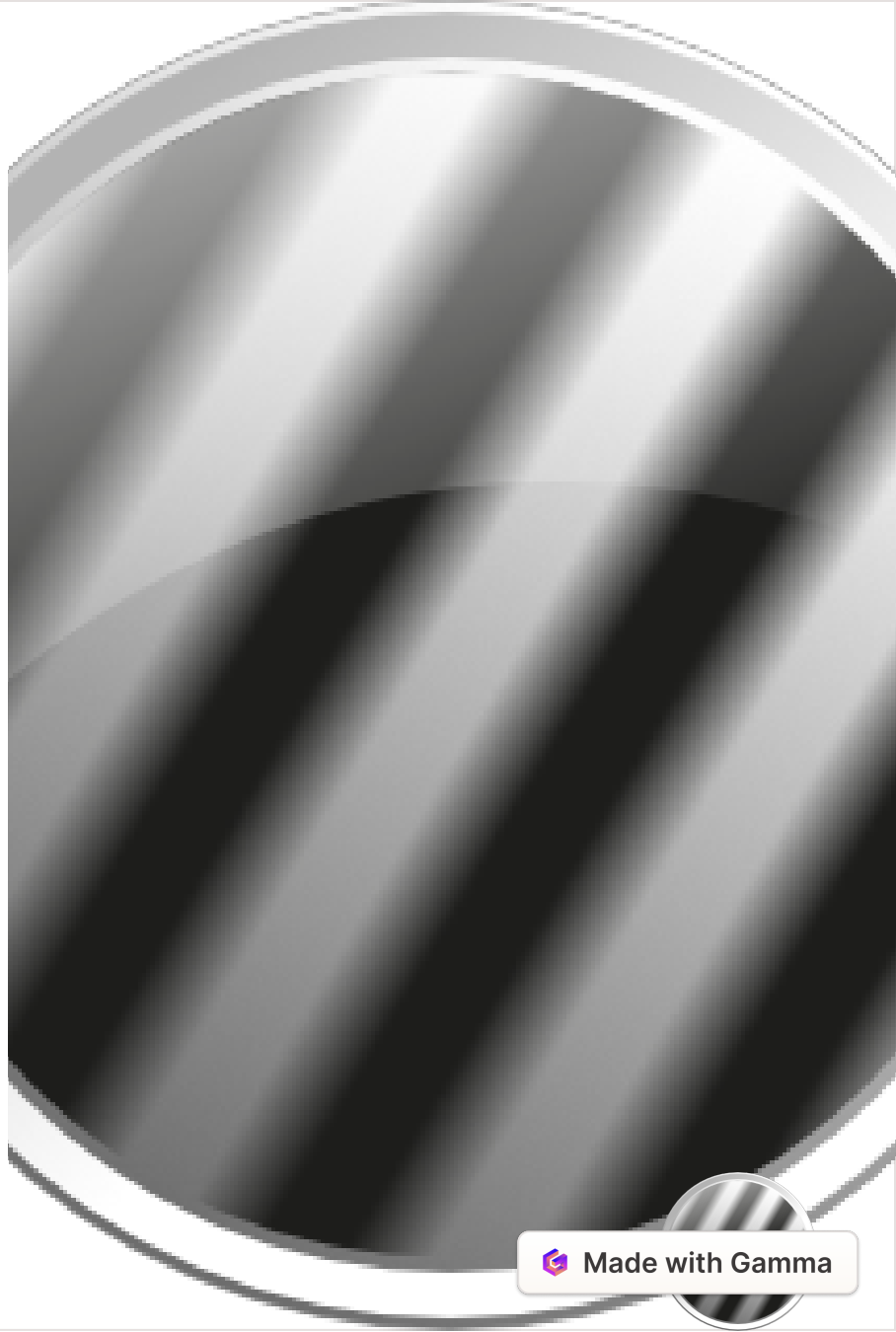
Creating PsychoPy Experiments to study Rodent Learning

In this session, we will explore the process of creating an experiment using PsychoPy, a powerful and user-friendly software for behavioural research. In this session, we will focus on paradigms used in animal learning and decision making.

by Becca Hirst



Materials

All materials for this workshop can be downloaded from **this google drive**.

This workshop was run in **PsychoPy 2023.2.3** Standalone version.

Session content

- Introduce our paradigm
- The basics of presenting stimuli and gathering responses.
- Using response input to dynamically update stimuli (make things move)
- Configuring sessions/blocks for experiments
- Breaking out of trial loops based on conditions (e.g. proportion correct).
- Reading input from serial devices (an Arduino Board).

Timetable

9:00 - 10:00 - Trouble Shoot Help

10:00 - 12:00 - Introducing creating Experiments

12:00 - 13:00 - Lunch

13:00 - 14:30 - Creating Experiments continued (adding code components)

14:30 - 15:00 - Break

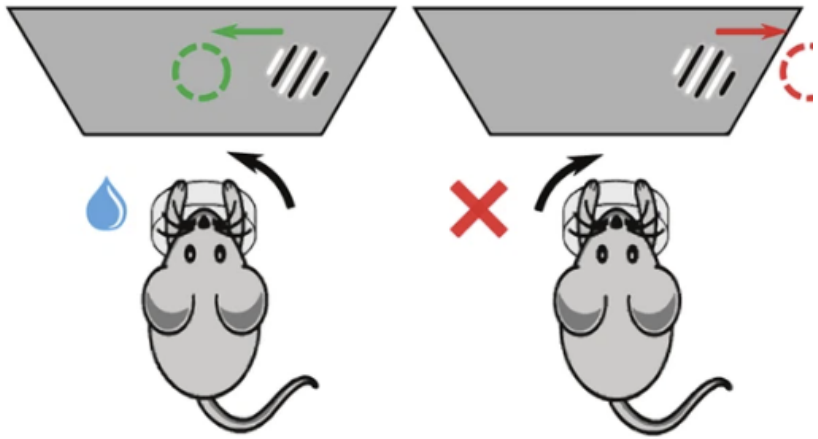
15:00 - 16:00 - Creating Experiments continued (reading a serial port)

16:00 - 17:00 - freestyle - apply your skills to create your own experiment.

The Task (basics)

Out task is *based** on [this paper] from The International Brain Laboratory.

**it is possible there are some adaptations for the purposes of this tutorial*



A two-alternative-forced-choice.

- Grating presented on left or right.
- Mice detect the presence of visual grating and respond by turning a steering wheel* to move the stimulus to the centre of the screen.
- Correct responses rewards a sugar drop, incorrect responses produce a noise burst.

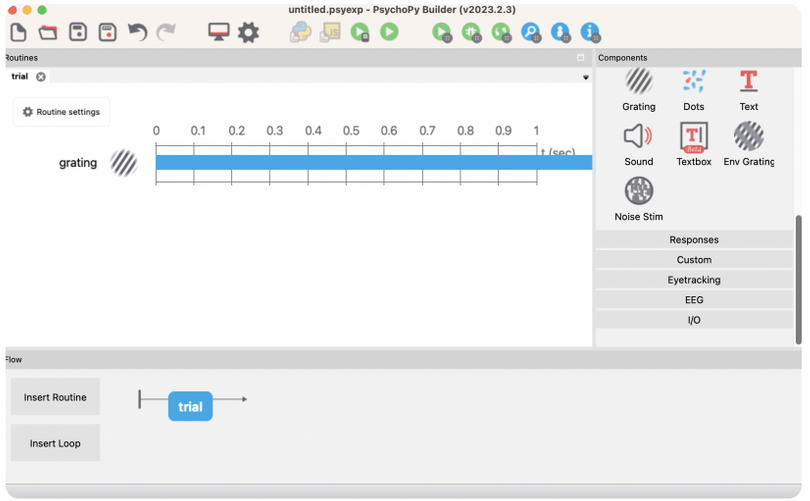
*We don't have a steering wheel, so we will start by simply polling mouse position to change stimulus location, and later use a serial port.

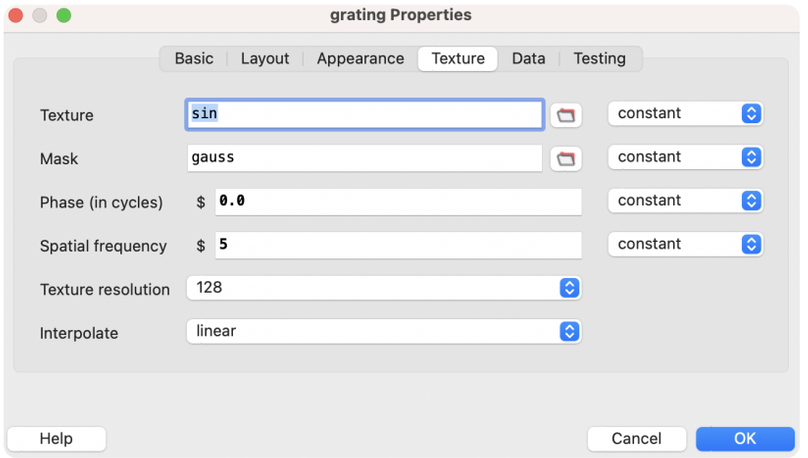
Present a grating

To get started open PsychoPy [Builder view](#).

The trial in our task contains a single stimulus, a [grating](#). Select grating from the components panel and configure the following parameters:

- Basic > Duration: blank
- Texture > Mask: gauss
- Texture > Spatial frequency: 5





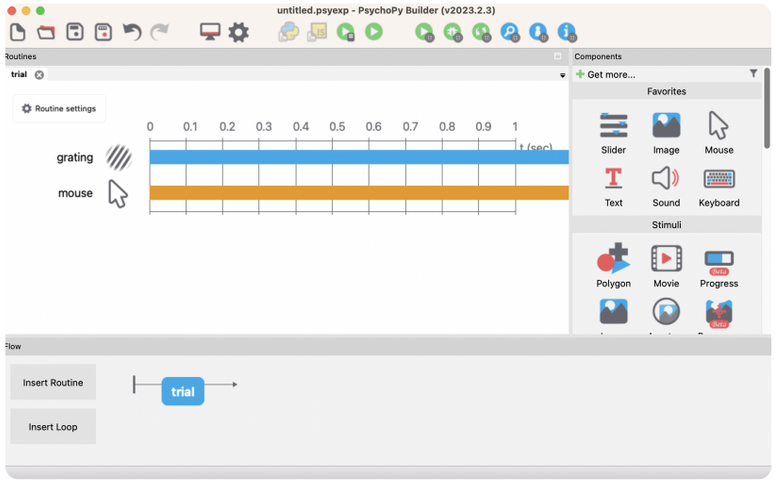
Gather a response

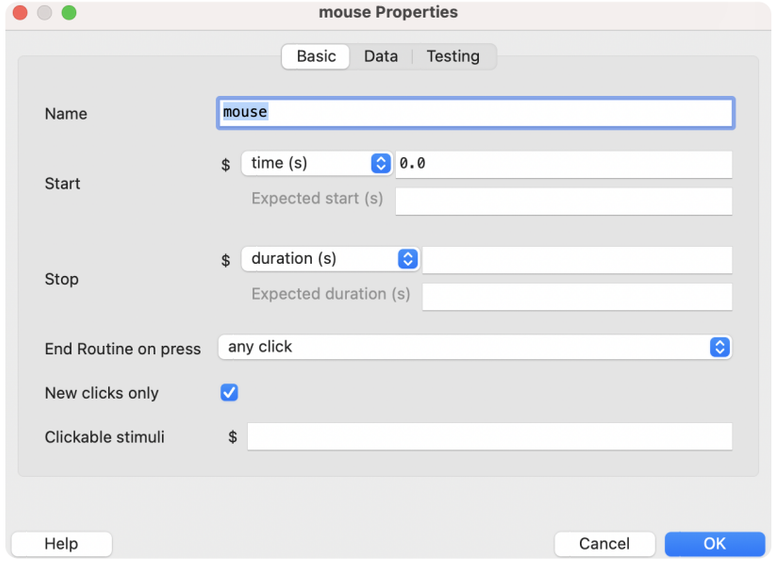
To start with we will use a mouse click as a response to end a trial.

Add a mouse component to your trial routine with the following parameters:

- Basic > Duration: Blank
- Basic > End Routine on press: any click (this means clicking the mouse will end the trial)

> You can store the mouse position frame-by-frame by setting "Save mouse state" to "each frame" .

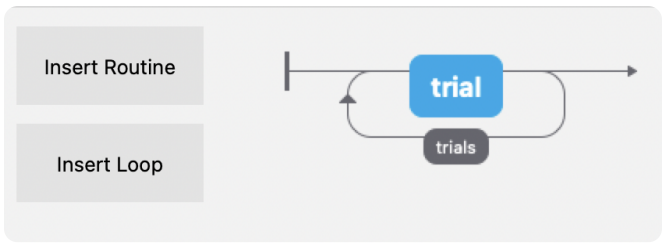




Update grating start position trial-by-trial

In the basic sessions of this task, the grating starts on the left or right side of the screen with equal probability. We therefore make a spreadsheet with an equal number of rows corresponding to the left position and right position.

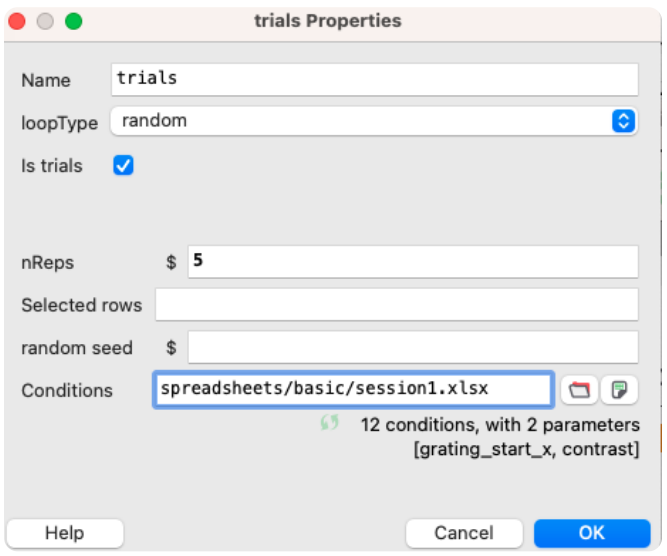
1 Add a loop around your trial routine.



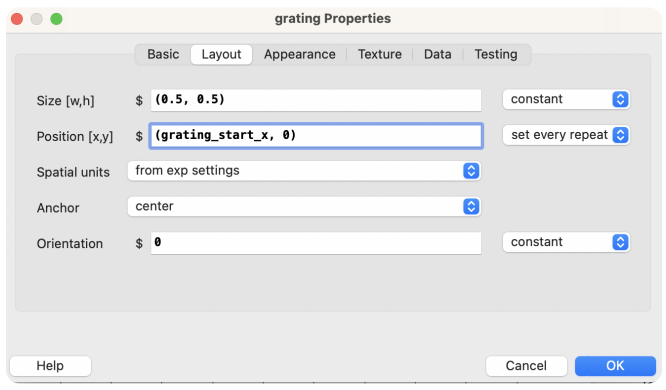
2 Configure a spreadsheet with an equal number of rows presenting stimuli on the left and right.

	A
1	grating_start_x
2	0.5
3	0.5
4	0.5
5	0.5
6	0.5
7	-0.5
8	-0.5
9	-0.5
10	-0.5
11	-0.5
12	

3 Set the "Conditions" field of your loop to be the path to your spreadsheet.



4 Update the position of your grating to use the column header from your spreadsheet. Set the field to "set every repeat"



By leaving **nReps** as 5, we repeat the 10 rows in our spreadsheet 5 times.

With **loopType** "random" the rows in our spreadsheet are presented in a random order, within each repetition.

Is trials means a new row will be created in our data output for each iteration of this loop.

Exercise

- Use your spreadsheet to update the grating contrast trial by trial. 50% of trials should be 100% contrast (1) and 50% should be 50% contrast (0.5).

Update grating position dynamically from input

Add a code component to your trial routine. In the Begin Routine tab add:

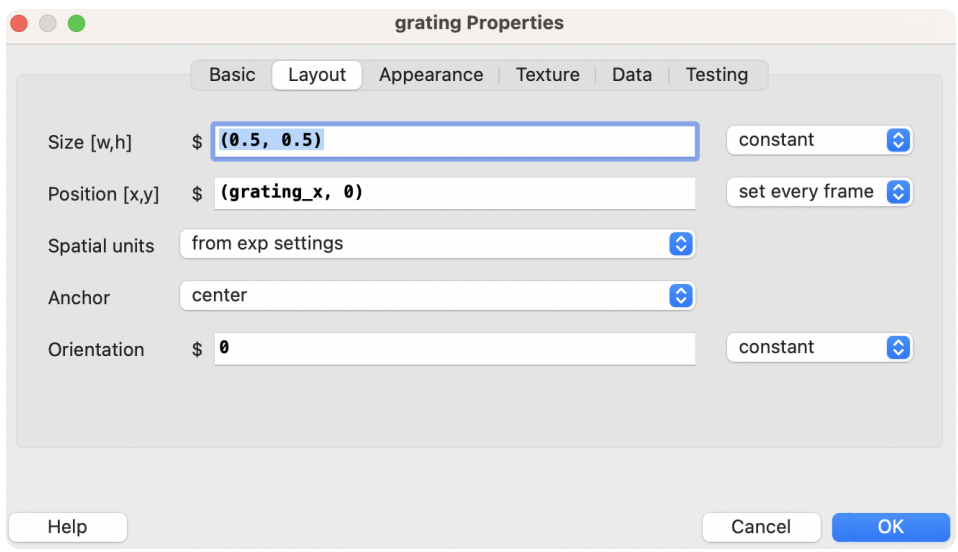
```
# set mouse position to middle of screen
mouse.setPos([0,0])

grating_x = grating_start_x
```

In the Each Frame tab add:

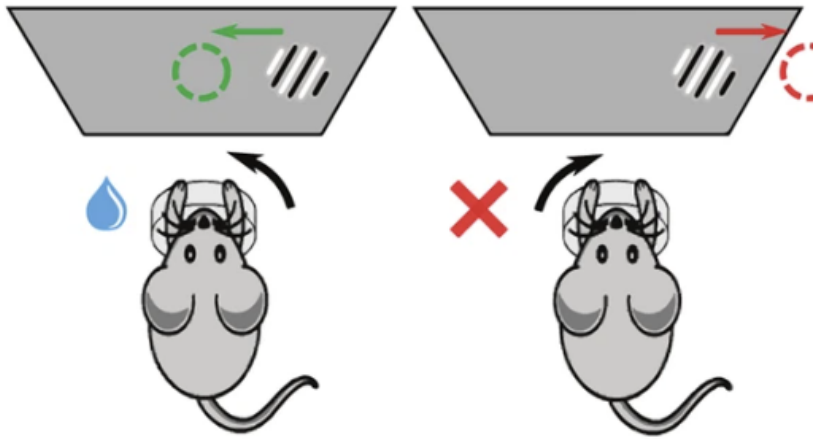
```
# update the x coordinate of the grating
grating_x =  grating_start_x + mouse.getPos()[0]
```

> The tab we use in code components corresponds to when we want code to be executed. In this case, we need to set the mouse position at the start of the trial, so we use Begin Routine (the begining of the trial routine). We then want to dynamically update a variable each time the screen refreshes, and so we use the Each Frame tab. Think about where on your flow/experiment you want code to be executed.

We can then use the value of `grating_x`, which is being updated every screen refresh based on mouse position, to dynamically change the location of the grating. In your grating component set Layout > Position to `(grating_x, 0)`and set the field to "set every frame".



The Task (basics)



A two-alternative-forced-choice.

- [x] ~~Grating presented on left or right.~~
- [ ] Mice detect the presence of visual grating and respond by turning a steering wheel to move the stimulus to the centre of the screen. **A mouse cannot click a cursor, we want to end the trial based on stim position.**
- [ ] Correct responses rewards a sugar drop, incorrect responses produce a noise burst.

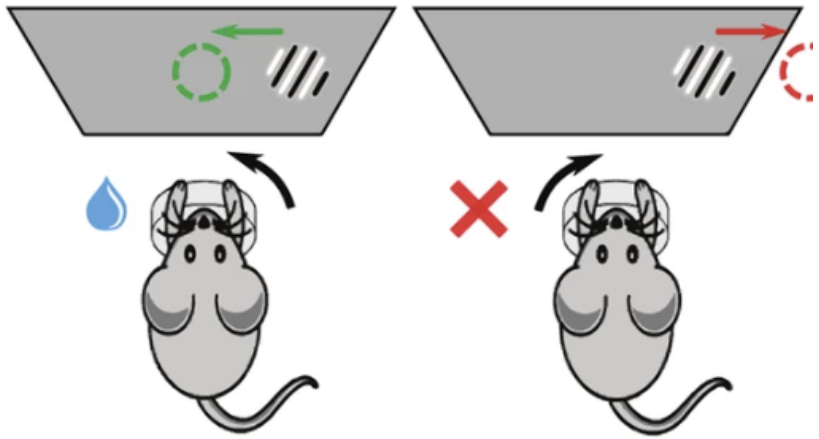
Ending a trial based on stimulus position

We want the trial to end when the grating either reaches the centre of the screen, or surpasses some threshold in the incorrect direction. Update your code component so that the "Each Frame" tab reads as:

```
# update the x coordinate of the grating
grating_x =  grating_start_x + mouse.getPos()[0]

# contingencies to end the routine
if grating_start_x < 0 and grating_x > 0:
   correct = 1
   continueRoutine = False
elif grating_start_x > 0 and grating_x < 0:
   correct = 1
   continueRoutine = False
elif grating_start_x < 0 and grating_x < -1:
   correct = 0
   continueRoutine = False
elif grating_start_x > 0 and grating_x > 1:
   correct = 0
   continueRoutine = False
```

The `continueRoutine` variable can be used to end a specific routine based on a criteria. Setting it to `False` will end the routine.

The Task (basics)



A two-alternative-forced-choice.

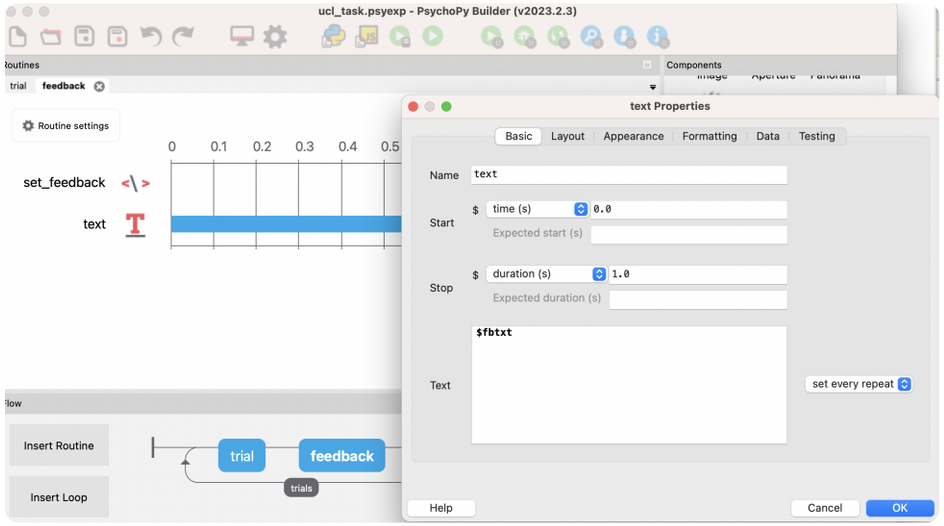
- [x] ~~Grating presented on left or right.~~
- [x] ~~Mice detect the presence of visual grating and respond by turning a steering wheel to move the stimulus to the centre of the screen.~~ ~~**A mouse cannot click a cursor, we want to end the trial based on stim position.**~~
- [ ] Correct responses rewards a sugar drop, incorrect responses produce a noise burst.

Adding trial-by-trial feedback

In our task we want to reward sugar for correct responses, or an auditory noise bust for incorrect responses. We don't have a sugar dispenser to hand (sadly), so we will instead start with some text to get started,

Add a new routine to your experiment called "feedback". Add a code component and in the Begin Routine tab add::

```
if correct:
    fbtxt = 'SUGAR'
else:
    fbtxt = 'NOISE'
```



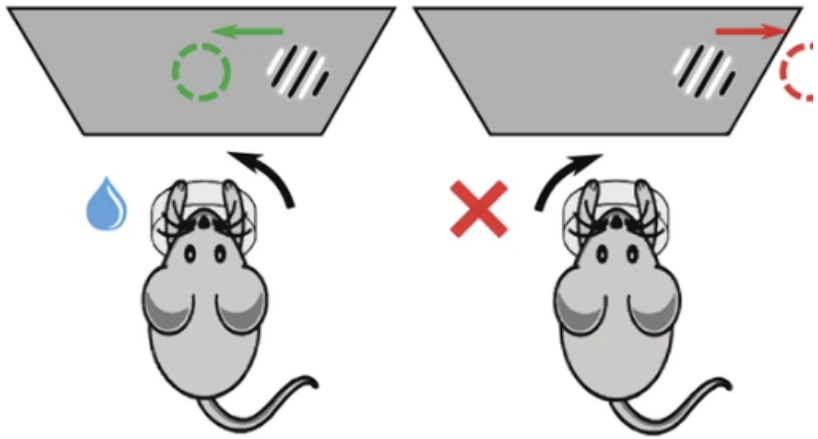
Then we use the variable `fbtxt` in the Text field of a Text component and set the field to "set every repeat".

> Use the `$` at the start of a field when using a variable or python code to set a parameter of a stimulus. You do not need a `$` if the field already has one.

Exercise - make some noise!

- Add a sound component. Set it's volume to be 0 if correct and 1 if correct. (if you have a wav file of some noise you can use that in the Sound field.

The Task (basics)

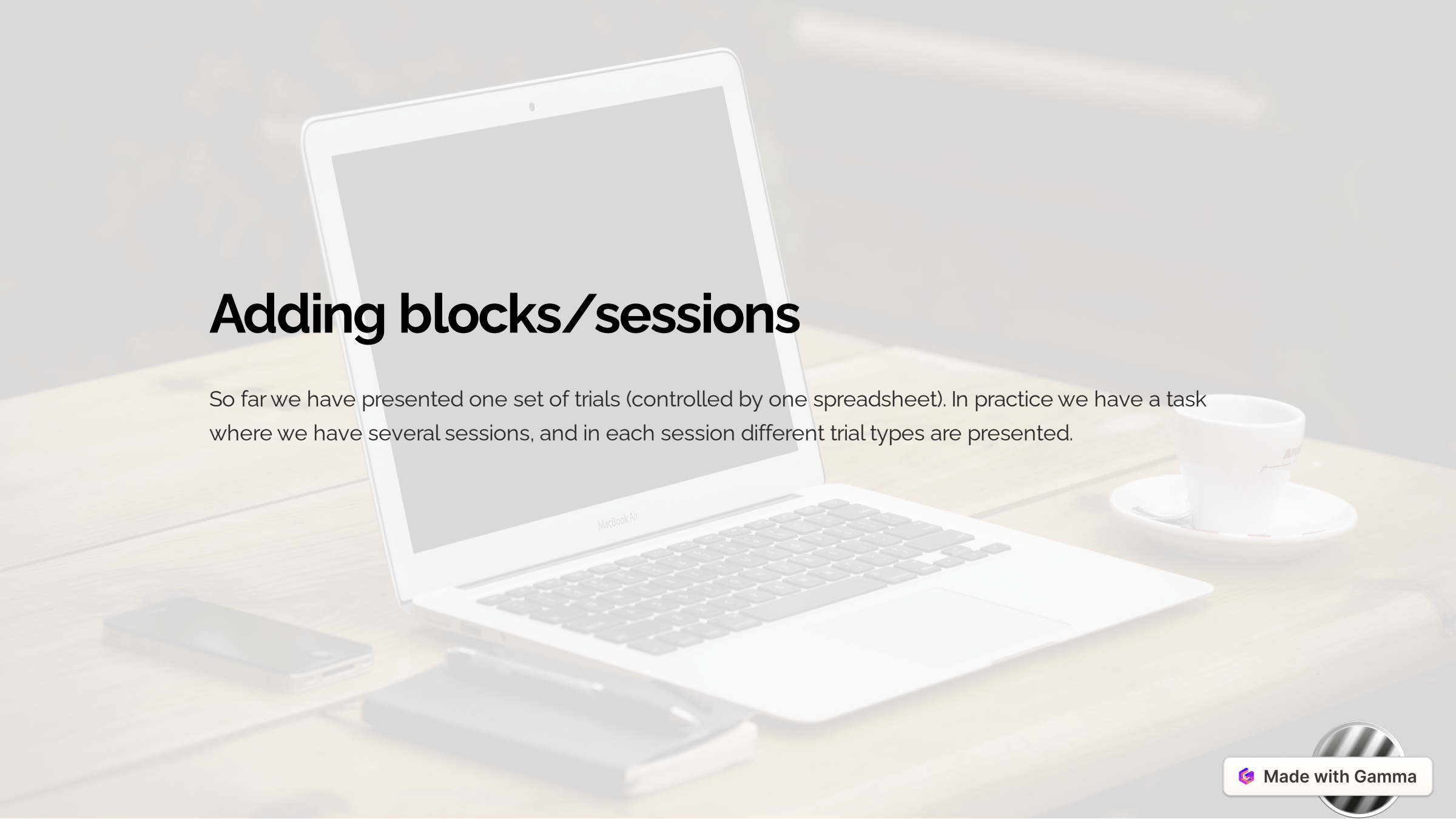


A two-alternative-forced-choice.

- ~~Grating presented on left or right.~~
- ~~Mice detect the presence of visual grating and respond by turning a steering wheel to move the stimulus to the centre of the screen.~~ **~~A mouse cannot click a cursor, we want to end the trial based on stim position.~~**
- ~~Correct responses rewards a sugar drop, incorrect responses produce a noise burst.~~

Adding blocks/sessions

So far we have presented one set of trials (controlled by one spreadsheet). In practice we have a task where we have several sessions, and in each session different trial types are presented.



Task blocks

In this task we have several sessions. Each session can vary in terms of left/right probability and stimulus contrast. The coloured text indicates what changes between sessions

Note: The proportion of trials for each stimulus contrast is adapted in this workshop from the original paper so that we can run fewer trials in total

1

Basic session

left/right probability: 50:50

stimulus contrasts: **100% or 50%**

2

Basic session

left/right probability: 50:50

stimulus contrasts: **50%, 25%**

3

Basic session

left/right probability: 50:50

stimulus contrasts: **25%, 12%**

4

Basic Session

left/right probability: 50:50

stimulus contrasts:**12%, 6%**

1

Full session

left/right probability: **20:80**

stimulus contrasts: 12%, 6%

2

Full session

left/right probability: **80:20**

stimulus contrasts: 12%, 6%

3

Full session

left/right probability: **20:80**

stimulus contrasts: 12%, 6%

4

Full session

left/right probability: **80:20**

stimulus contrasts: 12%, 6%

Setting up session trials

To get started we need to make a spreadsheet for each of the below.

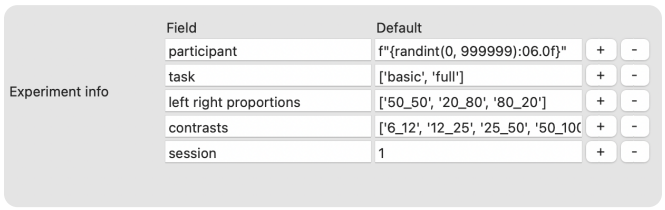
- Basic: 50/50 left/right contrast 50% or 100%
- Basic: 50/50 left/right contrast 50% or 25%
- Basic: 50/50 left/right contrast 25% or 12%
- Basic: 50/50 left/right contrast 12% or 6%
- Full task: 20/80 left/right contrast 12% or 6%
- Full task: 80/20 left/right contrast 12% or 6%

Note for ease all of these spreadsheets have been pre-created and can be found in the "spreadsheets" folder here. The `make_sheets.py` file is a helper script used to generate sheets quickly, it can be run in PsychoPy coder view.

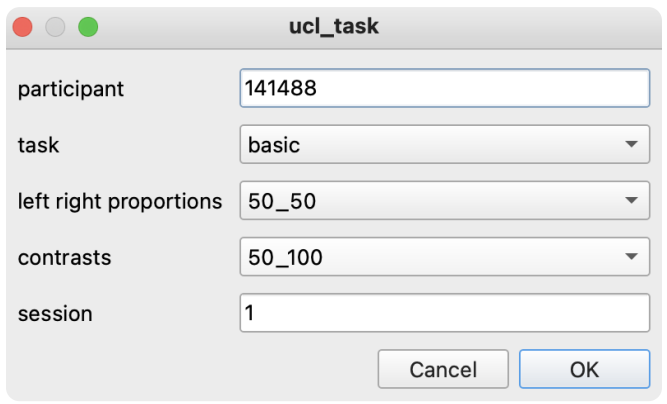
Calling different sheets in your experiment

We now have a different spreadsheet to load for each session, but how do we make it so that our experiment loads these different spreadsheets? Ideally, we want the researcher to be able to configure the session details at the start of the experiment, so that the correct file is selected.

1 Add a set of fields to the experiment startup dialogue to allow the user to use different sheets. (Select the cog icon to access experiment settings)

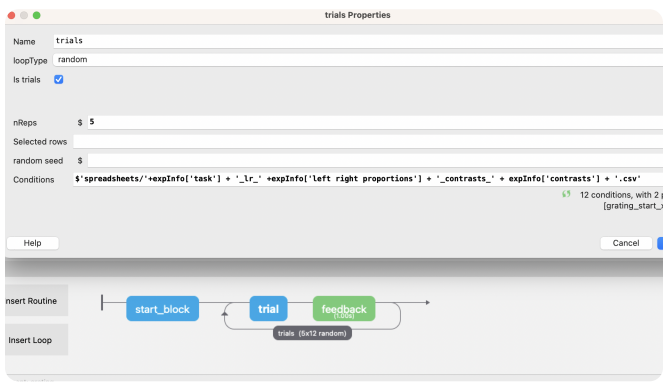


2 This will present drop down options to select different configurations at the start of your experiment.



3 Finally, update the "Conditions" field of your loop to use the values from the drop down to set the file name.

```
$'spreadsheets/'+expInfo['task'] + '_lr_' +expInfo['left right proportions'] + '_contrasts_' + expInfo['contrasts'] + '.csv'
```



Ending a loop based on conditions

By default, our task will present all trials in a spreadsheet, repeated by nReps as specified in the loop. But what if we want to end a series of trials early. This is particularly important in learning experiments, where trials often end based on some learning criteria, as opposed to a preset number of trials.

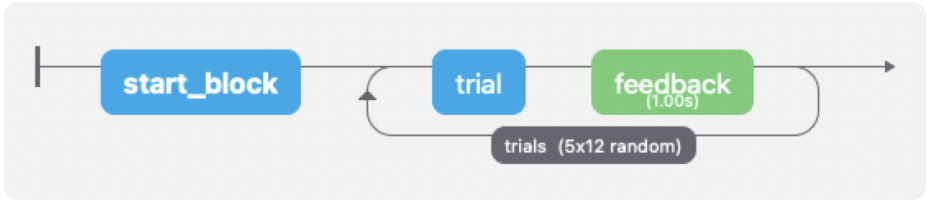
In our paradigm, we might want trials to end based on:

- **nTrials** e.g. when a max of 600 trials is presented
- **Duration of block/session** e.g. maximum 1 hour testing
- **Performance accuracy** e.g. when 80% accuracy is achieved

Ending a loop based on conditions

Ending a loop after set number of trials: If we have a max of 600 trials in a session, and our spreadsheet has 20 rows, set nReps to 30.

Ending a loop based on block duration: Add a routine to present before your loop starts.

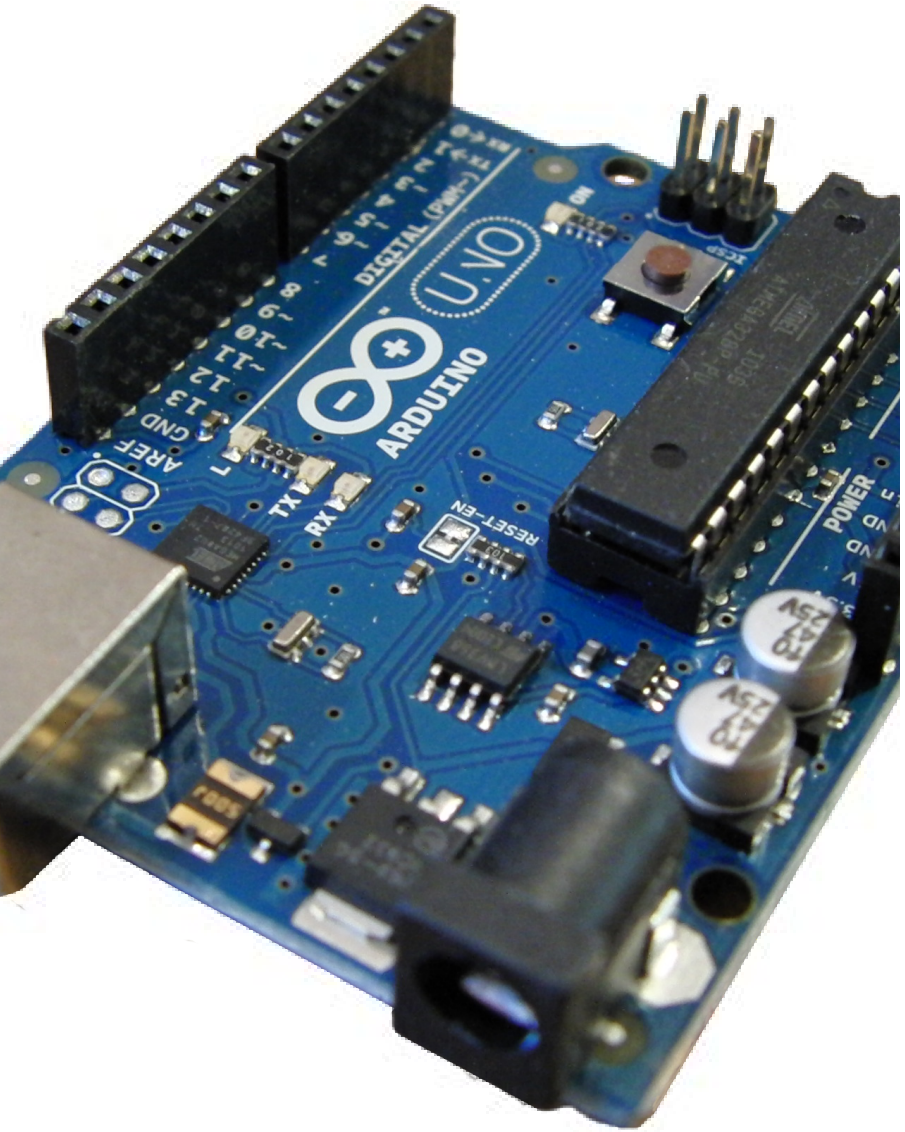


Add a code component to this routine and in the End Routine tab initialise a clock `blockClock = core.Clock()`. Then in your trial routine, you can continuously check the clock. in the Each Frame tab of any code component, use:

```
max_dur = 3600 # 1 hour max
if blockClock.getTime()> max_dur:
    continueRoutine = False
    trials.finished = True
```

Ending a loop based on proportion correct: Before the block of trials starts, initialise an empty list to track accuracy `acc = []` Then in the trial routine append this list with the value of `correct` using `acc.append(correct)`. Then we can use the following code snippet:

```
acc_thresh = 0.8 # proportion correct to end loop
min_trials = 10
prop_correct = sum(acc)/len(acc)
if trials.thisN > min_trials and prop_correct > acc_thresh:
    continueRoutine = False
    trials.finished = True
```



Getting input from external devices

In animal research, we often want the animal to respond using a specific external device. Most devices are connected via a serial port, so we need to know how to communicate with a serial device using PsychoPy,

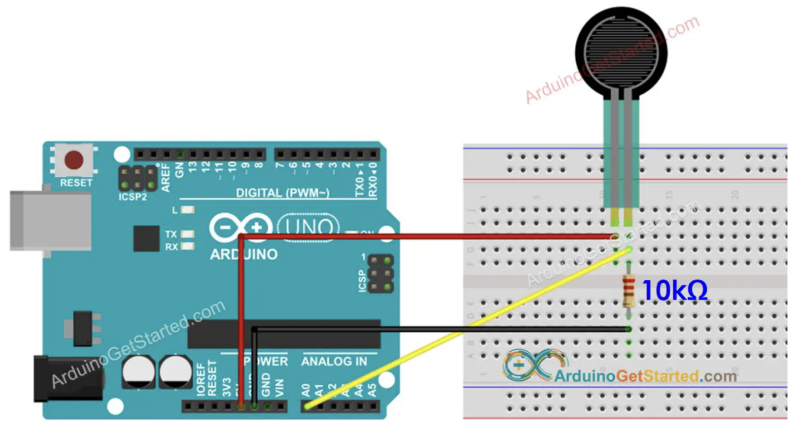
In this example we will walk through polling from a simple low cost arduino board. There are a wide range of sensors you can attach to arduino boards - but here we will use a force sensor.

Wire up your device (Arduino board)

First you need to set up your device. Here we follow **this guide** to set up our arduino board, which follows the wiring diagram on the right.

Then we open the Arduino IDE (can be downloaded for free **here**) and upload the code for reading from the device . The code we use is as below.

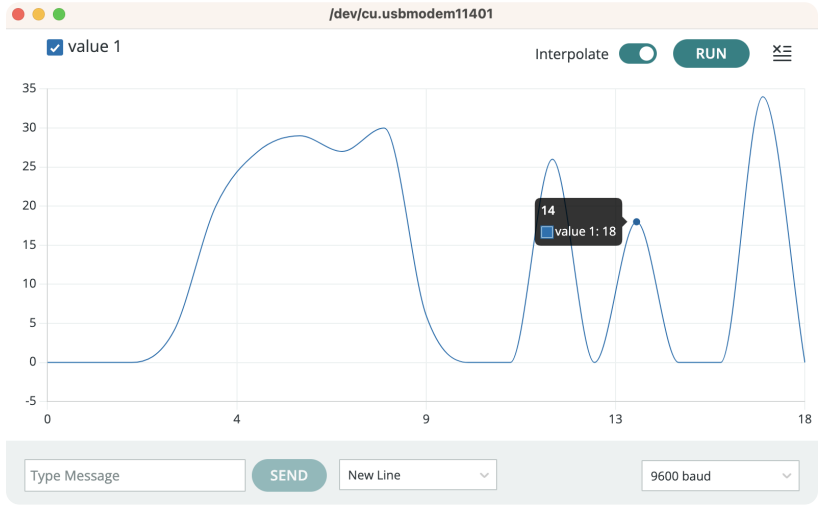
Wiring Diagram




```
const int forceSensorPin = A0; // Analog pin connected to the force sensor

void setup() {
  Serial.begin(9600); // Initialize serial communication
}

void loop() {
  int sensorValue = analogRead(forceSensorPin); // Read the analog input from the sensor
  Serial.println(sensorValue); // Print the sensor value to the serial monitor
  delay(1000); // Wait for a second (adjust as needed)
}
```

Wire up your device (Arduino board)

You can check that your device is providing input by opening Tools > Serial Monitor (or serial plotter) in the Arduino IDE, you should see a polled value printed to the output, that changes when you apply force.



Set up a serial port in PsychoPy

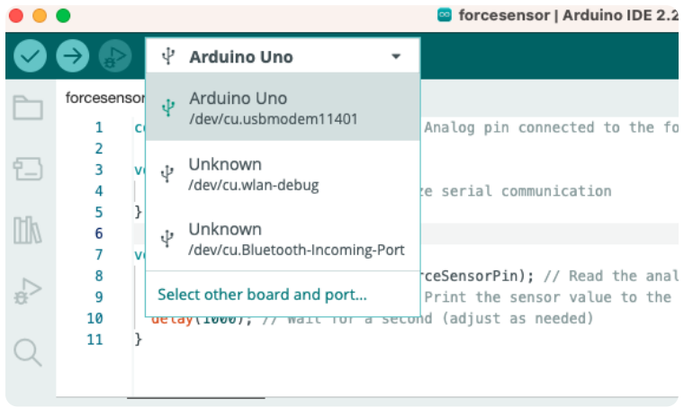
To read from a serial port in PsychoPy, add a code component to the first routine of your experiment. Import the relevant libraries in the Begin Experiment tab:

```
import serial
import time
```

Then configure your port in the End Routine tab:

```
port = serial.Serial('/dev/cu.usbmodem11401', 9600)
time.sleep(1) #Give the Arduino some time to wake up!
```

Here `'/dev/cu.usbmodem11401'` is the address of the serial port where the arduino is connected and 9600 is the baudrate. If you're not sure which port address your arduino is connected to, the Arduino IDE should tell you:

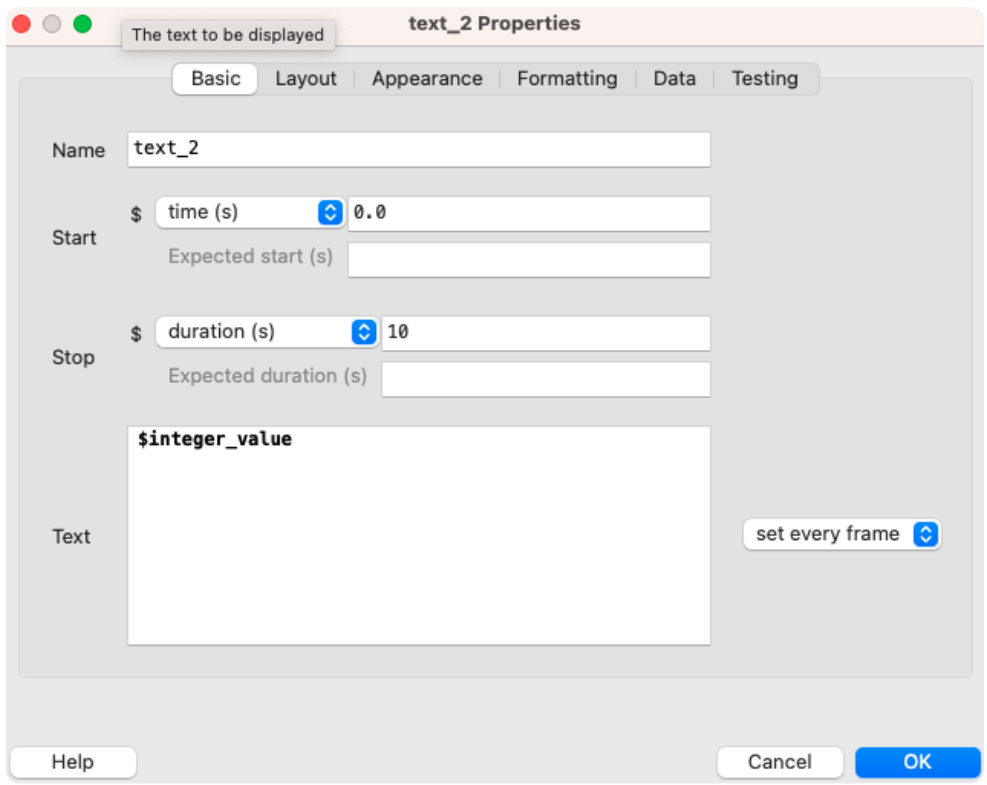


Read from a serial port in PsychoPy

To read from a serial port, we add a code component to our trial routine. If we want to poll the serial port dyanmically (e.g. to update a stimulus) we add code to the Each Frame tab:

```
#Read from the serial port
byte_string = port.readline()


integer_value = int(byte_string.decode('utf-8').strip())
```

Because the read value is in byte format. e.g. b'123\r\n' we need to convert it to an integer to use the returned serial value. To check this works, add a text component to your trial routine, set the Text field to be `$integer_value` and "set every frame".



⚠ It is important to close the Arduino IDE before attempting to run your PsychoPy experiment. Two pieces of software trying to communicate with the same port can cause errors.

Exercise

- Use the serial port value to change the position of a stimulus in PsychoPy

That's all for this session!

Useful resources for learning more about PsychoPy:

- Youtube Channel **psychopy_official**
- Textbook: **Creating Experiments in PsychoPy**
- Help forum: **discourse.psychopy.org**
- **Documentation**
- **API**
- Contribute to PsychoPy on **GitHub**!