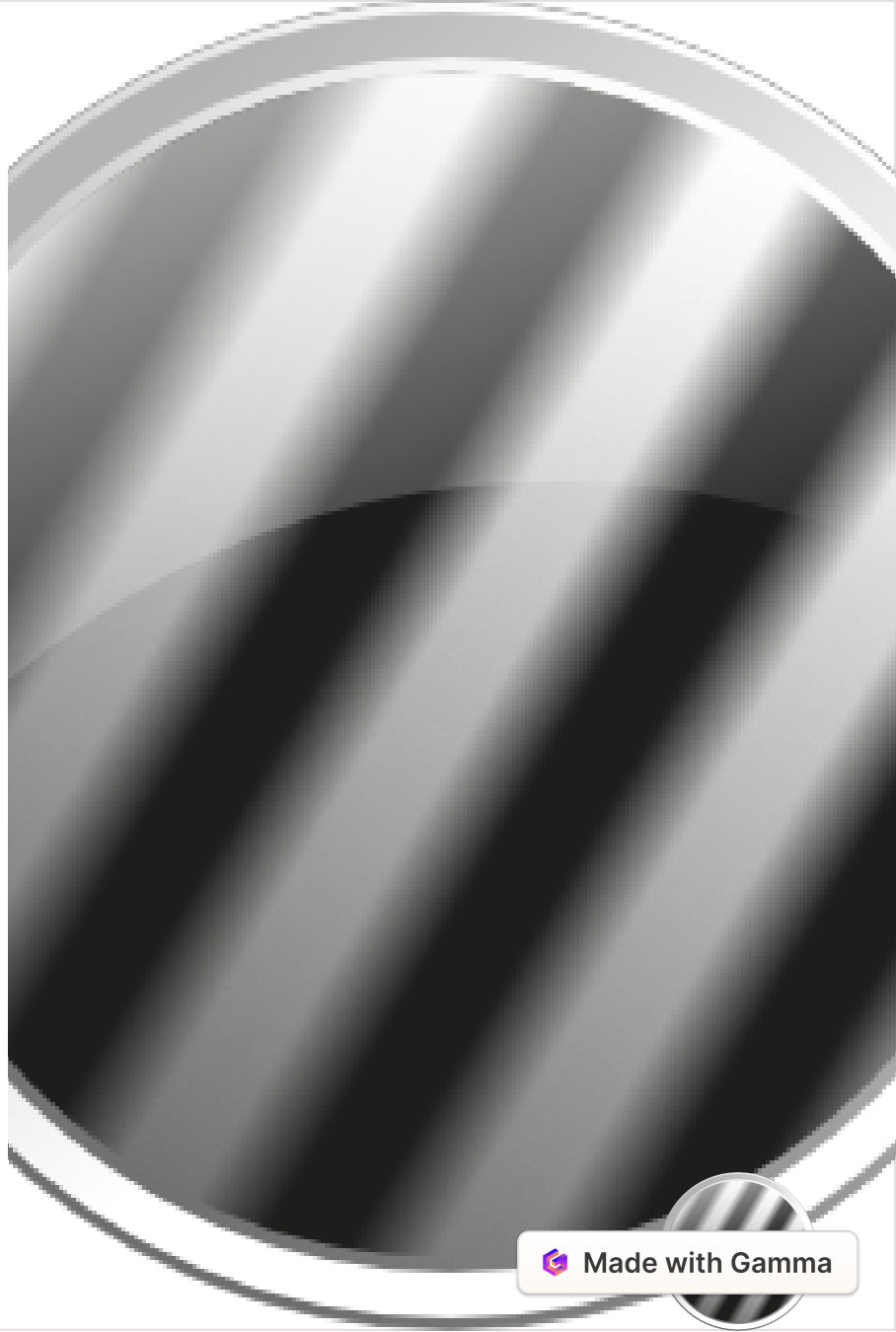
Creating an Experiment in Python

In this session, we will learn some python by coding an experiment using the PsychoPy library.

by Becca Hirst



Session content

- Creating and drawing stimuli.
- Gathering keyboard responses.
- Importing a trials spreadsheet and controlling them with TrialHandler.
- Saving the data.
- Presenting custom dialogue boxes.

This is similar to what we covered in session 1, except this time it will be entirely in python code.

Why code an experiment?

In our first session, we advised using the Builder interface to create experiments, so why bother learning to code one from scratch?

- Coding an experiment is a visual goal for learning coding
- Coding is a transferable skill
- You learn more about the PsychoPy library and what you can use in code components.

Starting your script

Start as you mean to go on, comment your code!

This is important so that future people (and you!) can use your code.

Then start by importing any PsychoPy modules you need.

```
'''
multiline comments
My first experiment

Author: Hirst RJ
'''


# single line comment
from psychopy import visual, data, core, gui
```

Create a window and stimuli

```
# create a window
win = visual.Window([1024,768], fullscr=False, units='pix')

# create a fixation
fixation = visual.Circle(win, size = 5,
    lineColor = 'white', fillColor = 'lightGrey')

# create a cue
cue = visual.ShapeStim(win,
    vertices = [[-30,-20], [-30,20], [30,0]],
    lineColor = 'red', fillColor = 'salmon')

# create a target
target = visual.GratingStim(win, size = 80, # 'size' is 3xSD for gauss,
    pos = [300, 0], #we'll change this later
    tex = None, mask = 'gauss',
    color = 'green')
```

Drawing stimuli

```
# draw the stimuli
fixation.draw()
win.flip()
core.wait(0.5)

cue.draw()
win.flip()
core.wait(0.5)

target.draw()
win.flip()
core.wait(1)
```

Understanding `Window.flip()`

- All the `draw()` commands operate on a memory buffer called the 'back buffer' on the graphics card.
- When you `flip()` the window it causes everything in that 'back buffer' to become visible on the physical screen.
- The flip() command waits until the next screen refresh to present your stimuli (every 1/60s, so about 16.6ms)
- It will then wait until the physical screen refresh occurs (if possible with your graphics card settings)

This has some implications...

- That means your screen flips (and intervening code) are tied to a fixed rate of 1/60s
- It is physically impossible to draw your stimulus for partial frames (e.g. 25ms) on a 60Hz screen
- Also, if Python/PsychoPy has to run too much code between flips you might 'drop' a frame (fail to get it drawn by the time of the screen refresh)
- If you don't call `flip()` for a while, or if you drop a frame, the screen will stay as it is for another frame

> To check how reliable your frame rate is. Open PsychoPy coder, select "Demos > timing > timeByFrames.py" this will show you a frequency distribution of the recorded frame intervals. On a 60Hz monitor, you would want a tight normal distribution around 16.66ms.

Exercise

- In PsychoPy Builder, create a text stimulus that says "Hello world".
- Compile it to python code.
- In the compiled python code find where the text stimulus is created.
- Copy that code and bring it into your coded experiment, try to draw it.

Waiting for a keypress response

Ideally, we want the target to remain onscreen until a keypress response is made.

```
from psychopy.hardware import keyboard

# create a keyboard object
kb = keyboard.Keyboard()

...

target.draw()
win.flip()

# wait for a keypress response
keys = kb.waitKeys(keyList=['left', 'right'])
print(keys[-1].name)# print the name of the keypress
print(keys[-1].rt)# print the response time of the keypress
```

Response times from keypress responses

We need to reset the timer on the clock, but we want the 0 point to be when the target is visible, for this reason we want to start the clock when the window flips using `win.callOnFlip`

```
target.draw()

#reset the keyboard clock to 0 when the target is visible
win.callOnFlip(kb.clock.reset)
win.flip()

# wait for a keypress response
keys = kb.waitKeys(keyList=['left', 'right'])
print(keys[-1].name)# print the name of the keypress
print(keys[-1].rt)# print the response time of the keypress
```

Presenting several trials

To present a series of trials we need to repeat our stimulus presentation, keyboard response within a loop. We could just use a `for` loop:

```
for trial in range(5):
    # draw the stimuli
    fixation.draw()
    win.flip()
    core.wait(0.5)

    cue.draw()
    win.flip()
    core.wait(0.5)

    target.draw()

    #reset the keyboard clock to 0 when the target is visible
    win.callOnFlip(kb.clock.reset)
    win.flip()

    # wait for a keypress response
    keys = kb.waitKeys(keyList=['left', 'right'])
    print(keys[-1].name)# print the name of the keypress
    print(keys[-1].rt)# print the response time of the keypress
```

But this doesn't give us all the flexibility we need, ideally we want to work through a series of trials where the stimuli update (like how PsychoPy Builder works!), for this we need to use PsychoPy's `TrialHandler` (read more info **here**).

TrialHandler

To use trial handler, you first need to set up a spreadsheet like we did for PsychoPy Builder, one column per variable (`target_x`, `cue_orientation`, `correct_keypress`) - we can recycle the spreadsheet we made from session 1. Here we will learn how PsychoPy Builder imports that spreadsheet:

```
# import spreadsheet
conditions = data.importConditions('conditions.xlsx')
print(conditions)

# give conditions to a trial handler object
trials = data.TrialHandler(trialList=conditions, nReps=5, method = 'random')
```

By using `print(conditions)` you can see what the spreadsheet is "under the hood", it appears as a list of dictionaries:

`[{'target_x':-0.5, 'cue_orientation':90, 'correct_keypress':'left'}, {'target_x':0.5, 'cue_orientation':90, 'correct_keypress':'right'}, {'target_x':-0.5, 'cue_orientation':270, 'correct_keypress':'left'}, {'target_x':0.5, 'cue_orientation':270, 'correct_keypress':'right'}]`

Presenting several trials

This time we can iterate through the list of dictionaries that we imported and randomised using the trial handler. In the loop below `trial` corresponds to a single dictionary from the `trials` object:

```
for trial in trials:# use the trial handler object (the list of dicts)

    # update stimuli using the "trial" dictionary
    cue.setOri(trial['cue_orientation']
    target.setPos([trial['target_x'], 0])

    ...
```

Storing the data - Experiment Handler

To store the data we can use PsychoPys **Experiment Handler** you can set up your Experiment Handler as below:

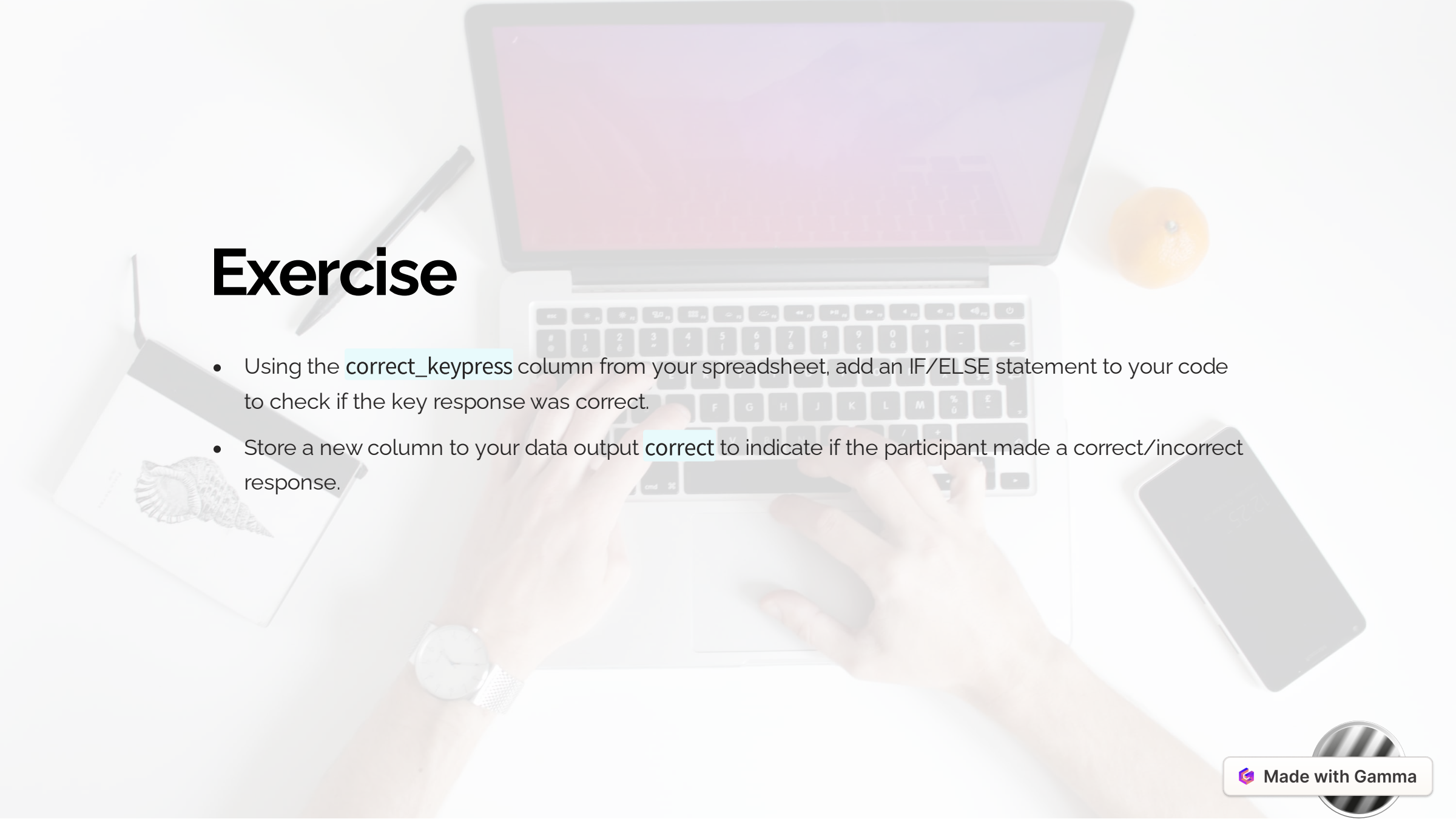
```
# import spreadsheet
conditions = data.importConditions('conditions.xlsx')
print(conditions)

# give conditions to a trial handler object
trials = data.TrialHandler(trialList=conditions, nReps=5, method = 'random')

# set up an experiment handler object
thisExp = data.ExperimentHandler(
    name='Posner', version='1.0', #not needed, just handy
    extraInfo = info, #the info we created earlier
    dataFileName = 'output', # a file name for the data output
    )
thisExp.addLoop(trials) #give the trials to the experiment
```

We then need to wait until after a response is made to store the data:

```
...

  #reset the keyboard clock to 0 when the target is visible
  win.callOnFlip(kb.clock.reset)
  win.flip()

# wait for a keypress response
keys = kb.waitKeys(keyList=['left', 'right'])
print(keys[-1].name)# print the name of the keypress
print(keys[-1].rt)# print the response time of the keypress

# save to the data file
thisExp.addData('key_response', keys[-1].name)
thisExp.addData('response_time', keys[-1].rt)

# move onto next row in data file
thisExp.nextEntry()
```

Exercise

- Using the `correct_keypress` column from your spreadsheet, add an IF/ELSE statement to your code to check if the key response was correct.
- Store a new column to your data output `correct` to indicate if the participant made a correct/incorrect response.



Add a startup dialogue box

A startup gui is useful to allow researchers to change parameters of your experiment, as well as gather important info at the start of your experiment (e.g. participant ID, session number). In PsychoPy Builder we access this in a "dictionary" known as `expInfo` so let's set that up the same in code:

```
from psychopy.hardware import keyboard

# create a keyboard object
kb = keyboard.Keyboard()

# a dictionary with fields you want to present in your dialogue
expInfo = {'participant':'', 'session':'', 'fixTime':0.5, 'cueTime':0.5}

# present dialog
dlg = gui.DlgFromDict(info) #(and from psychopy import gui at top of script)

if not dlg.OK:# if user does not press OK quit
    core.quit()

# you can also add parameters after the dialogue
info['dateStr'] = data.getDateStr() #will create str of current date/time

# you can use a formatted string to create a unique filename
filename = "{participant}_{dateStr}".format(**info)
```

You can then give this info to your experiment handler, so that it is automatically saved, and so that is uses a custom filename for data storage:

```
# set up an experiment handler object
thisExp = data.ExperimentHandler(
    name='Posner', version='1.0', #not needed, just handy
    extraInfo = expInfo, # info from the start dlg
    dataFileName = filename, # use the new formatted filename
    )
```

The whole code

At this stage, your whole code should look like this, note that we've updated this to also use `expInfo['fixTime']` and `expInfo['cueTime']` from your startup dialogue to control the timing of your stimuli from the beginning:

```
'''
multiline comments
My first experiment

Author: Hirst RJ
'''

# single line comment
from psychopy import visual, data, core, gui

# create a window
win = visual.Window([1024,768], fullscr=False, units='pix')

# a dictionary with fields you want to present in your dialogue
expInfo = {'participant':'', 'session':'', 'fixTime':0.5, 'cueTime':0.5}

# present dialog
dlg = gui.DlgFromDict(info) #(and from psychopy import gui at top of script)

if not dlg.OK:# if user does not press OK quit
  core.quit()

# you can also add parameters after the dialogue
info['dateStr'] = data.getDateStr() #will create str of current date/time

# you can use a formatted string to create a unique filename
filename = "{participant}_{dateStr}".format(**info)

# create a fixation
fixation = visual.Circle(win, size = 5,
    lineColor = 'white', fillColor = 'lightGrey')

# create a cue
cue = visual.ShapeStim(win,
    vertices = [[-30,-20], [-30,20], [30,0]],
    lineColor = 'red', fillColor = 'salmon')

# create a target
target = visual.GratingStim(win, size = 80, # 'size' is 3xSD for gauss,
    pos = [300, 0], #we'll change this later
    tex = None, mask = 'gauss',
    color = 'green')

# import spreadsheet
conditions = data.importConditions('conditions.xlsx')
print(conditions)

# give conditions to a trial handler object
trials = data.TrialHandler(trialList=conditions, nReps=5, method = 'random')

# set up an experiment handler object
thisExp = data.ExperimentHandler(
    name='Posner', version='1.0', #not needed, just handy
    extraInfo = info, #the info we created earlier
    dataFileName = 'output', # a file name for the data output
    )
thisExp.addLoop(trials) #give the trials to the experiment

for trial in trials:

  # update stimuli using the "trial" dictionary
  cue.setOri(trial['cue_orientation']
  target.setPos([trial['target_x'], 0])

  # draw the stimuli
  fixation.draw()
  win.flip()
  core.wait(expInfo['fixTime'])

  cue.draw()
  win.flip()
  core.wait(expInfo['cueTime'])

  target.draw()

  #reset the keyboard clock to 0 when the target is visible
  win.callOnFlip(kb.clock.reset)
  win.flip()

  # wait for a keypress response
  keys = kb.waitKeys(keyList=['left', 'right'])
  print(keys[-1].name)# print the name of the keypress
  print(keys[-1].rt)# print the response time of the keypress

  # save to the data file
  thisExp.addData('key_response', keys[-1].name)
  thisExp.addData('response_time', keys[-1].rt)

  # move onto next row in data file
  thisExp.nextEntry()
```

Accurate timing of stimuli

So far, we've used `core.wait(time)` to control how long we present stimuli for. It is notable that for most precise control over stimulus timing, you might want to use frames as a unit rather than seconds. This is done automatically by PsychoPy Builder, but it is a bit trickier to implement, so we did not use it in out code yet. To present stimuli in terms of frames you would make the following replacements:

```
# draw the stimuli
  fixation.draw()
  win.flip()
  core.wait(expInfo['fixTime'])

  cue.draw()
  win.flip()
  core.wait(expInfo['cueTime'])
```

Would be replaced instead with:

```
# draw the stimuli
  for frameN in range(fixFrames):
    fix.setAutoDraw(True)
    win.flip()
  fix.setAutoDraw(False)

  for frameN in range(cueFrames):
    cue.setAutoDraw(True)
    win.flip()
  cue.setAutoDraw(False)
```

The `setAutoDraw` function will keep drawing a stimulus on every frame until specified (i.e. until `setAutoDraw(False)`. Note you would also need to set up the values for `fixFrames` and `cueFrames` - taking into account the refresh rate of your monitor.

That's all for this session!