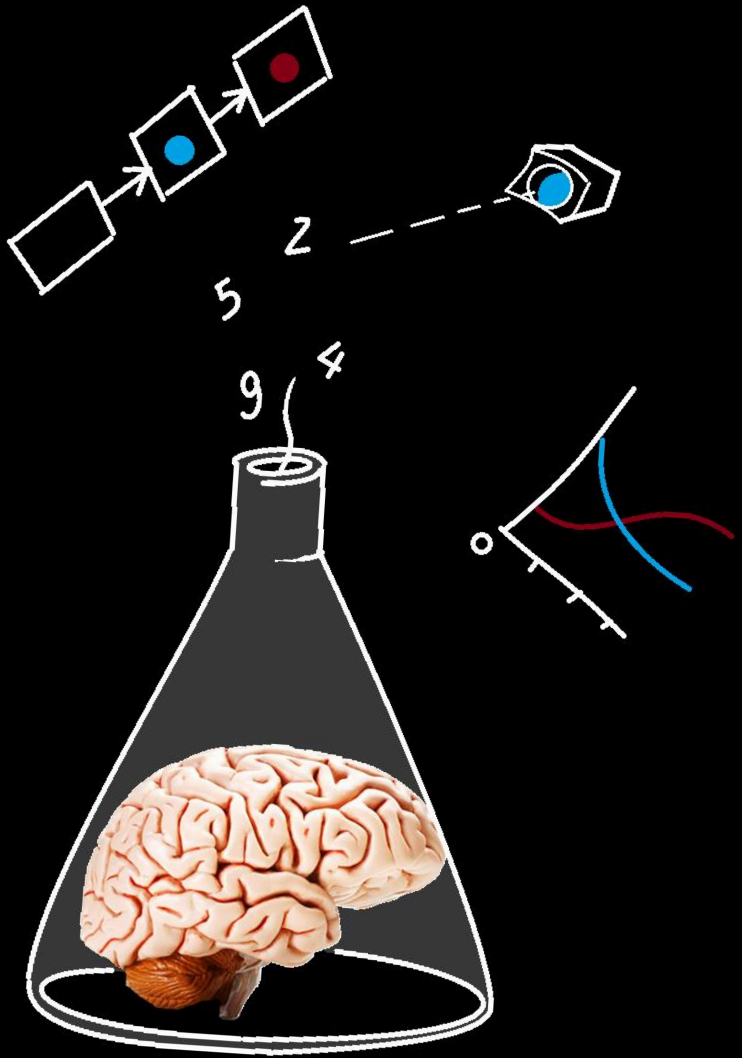
25-09

# Eye-tracking and pupillometry



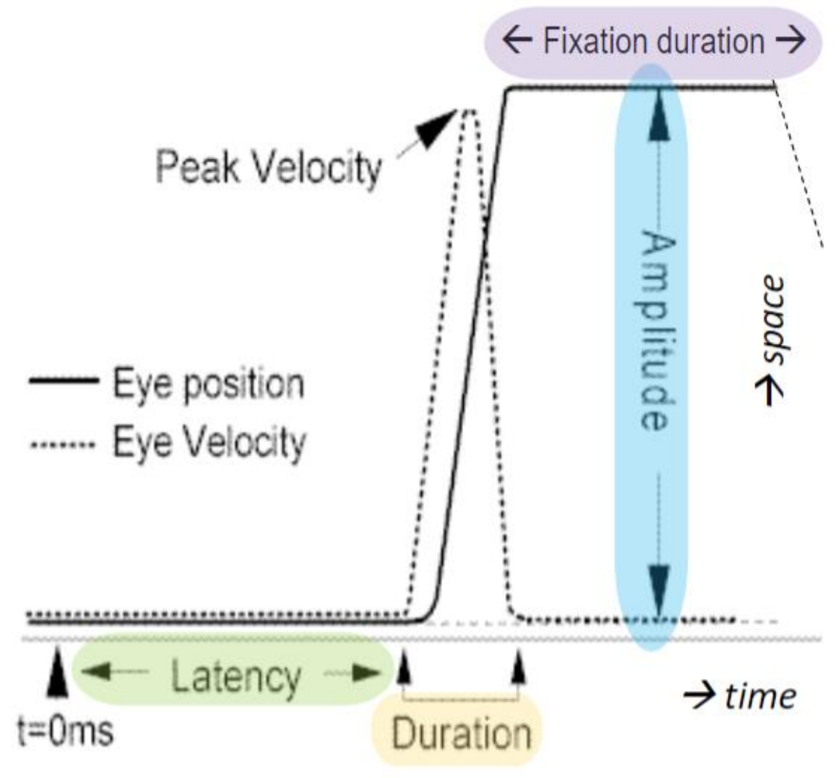
# When and why do we need eye-tracking?

Descriptive research: we’re interested in the what, why and where of eye movements themselves

Explanatory research: oculomotor data may provide a window onto various cognitive processes

# Terminology

Saccade
Saccadic amplitude
Saccadic latency
Fixation
Fixation duration
Microsaccade



# Terminology

Saccade
Saccadic amplitude
Saccadic latency
Fixation
Fixation duration
Microsaccade

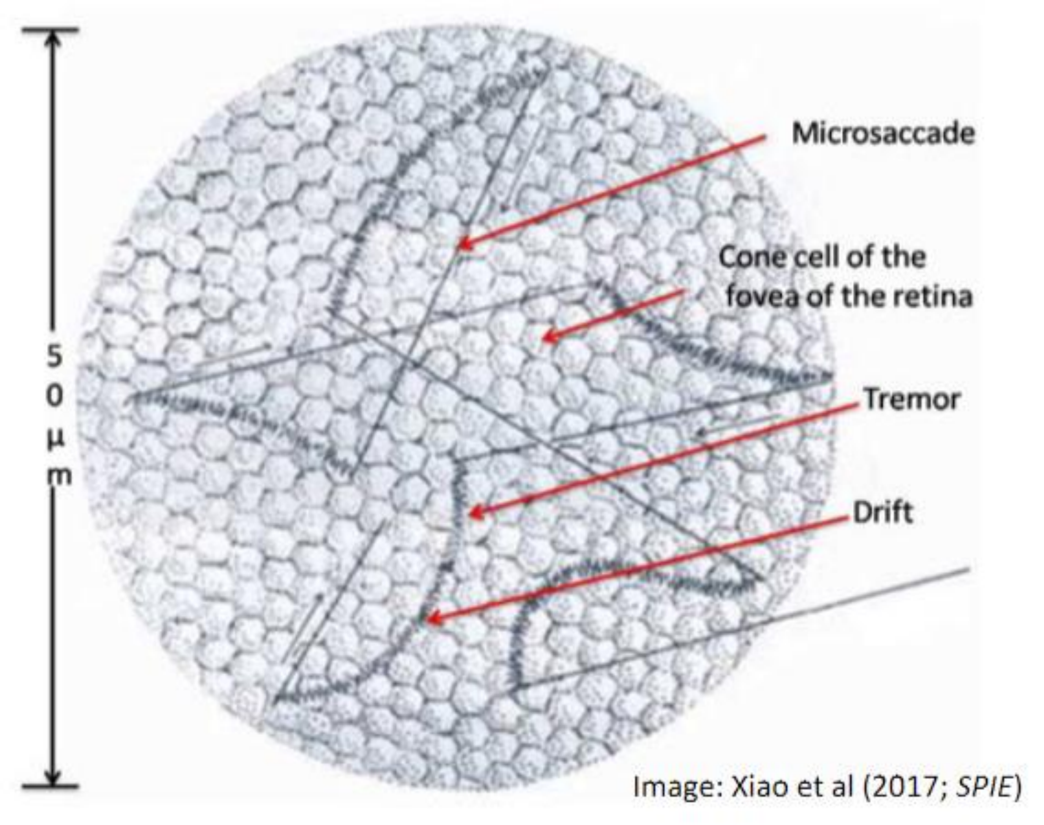
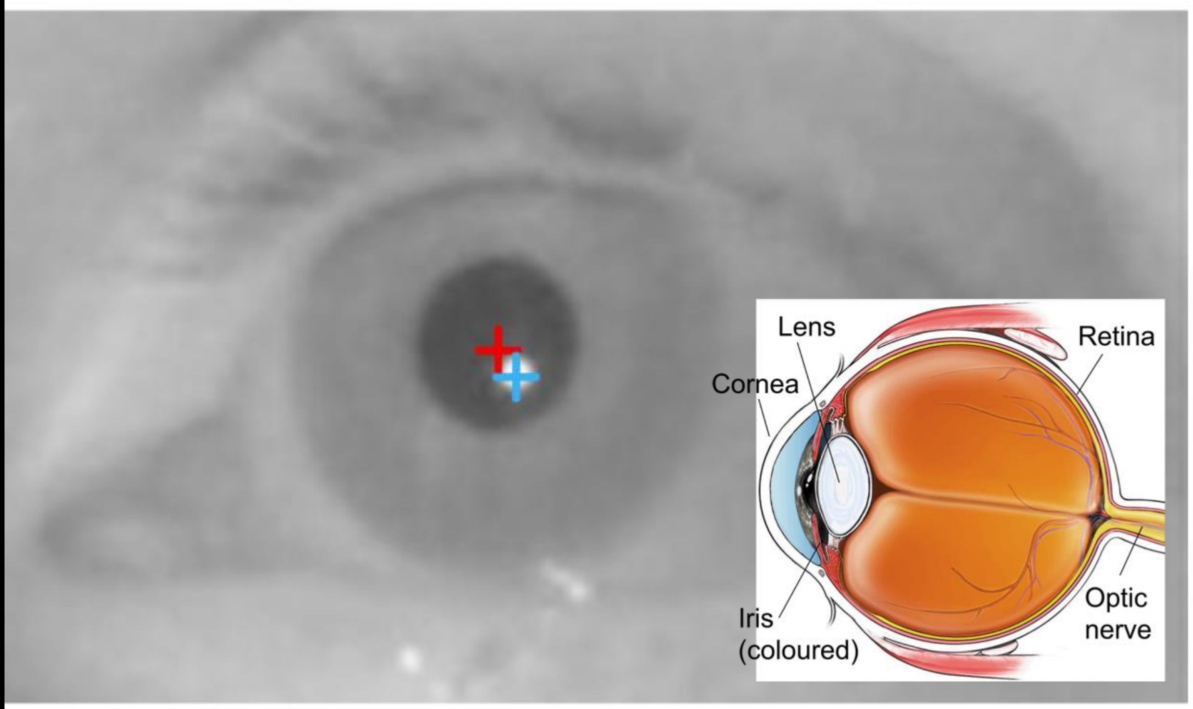
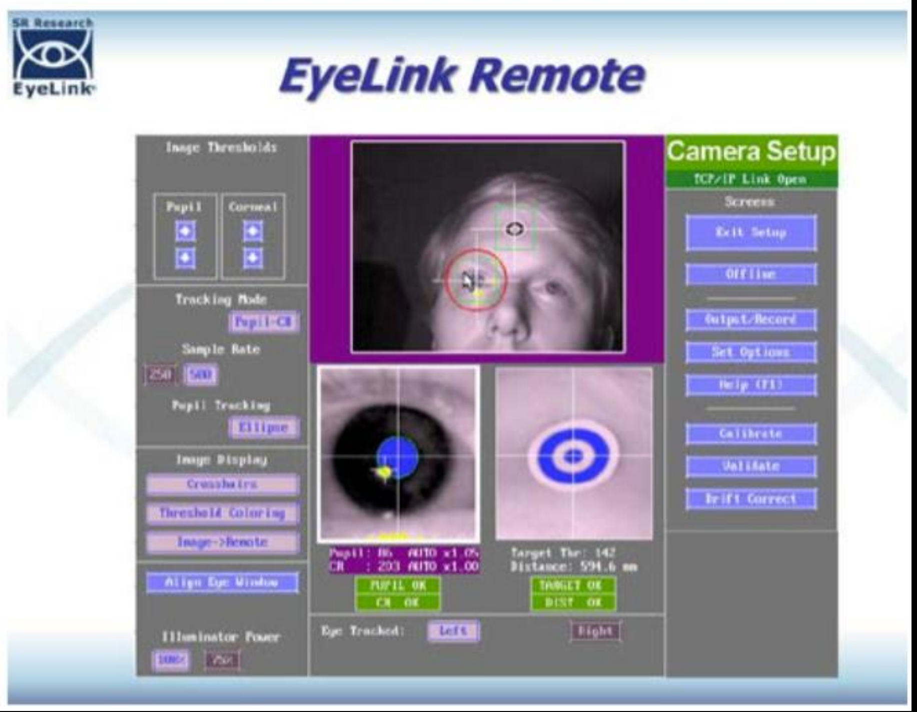


Image: Xiao et al (2017; *SPIE*)

# Eye position: two signals





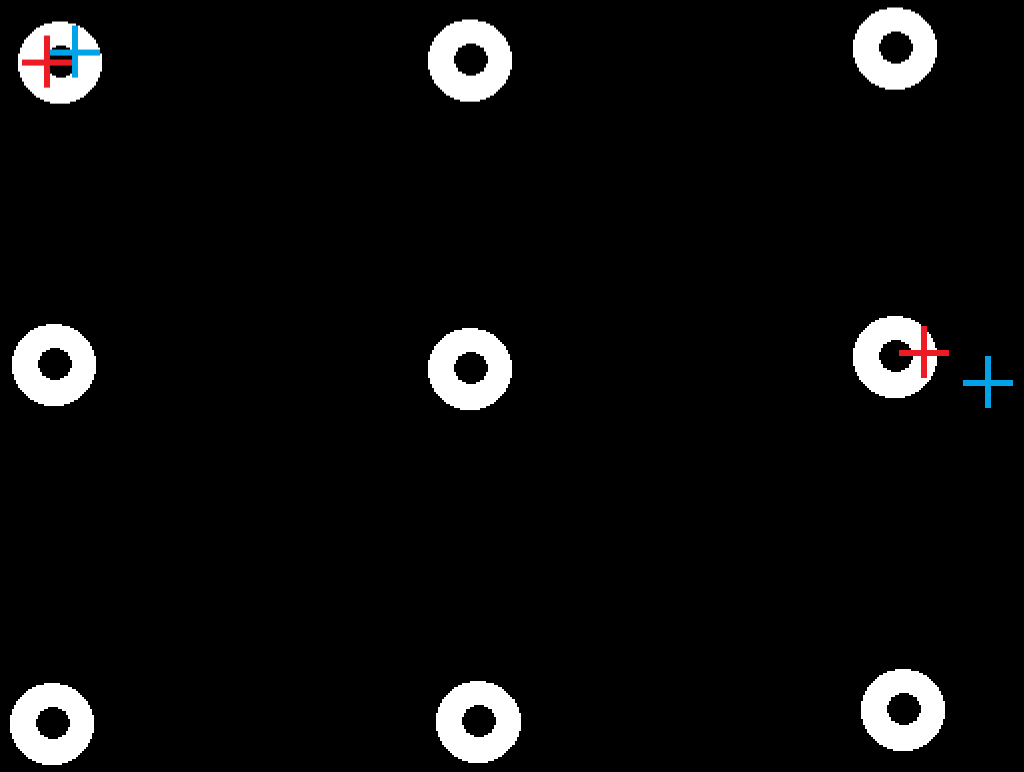
Pupil: 86 AUTO x1.05
CR : 203 AUTO x1.00

Target Thr: 142
Distance: 594.6 mm

Pupil location

Corneal reflection of (infrared) light sent from camera

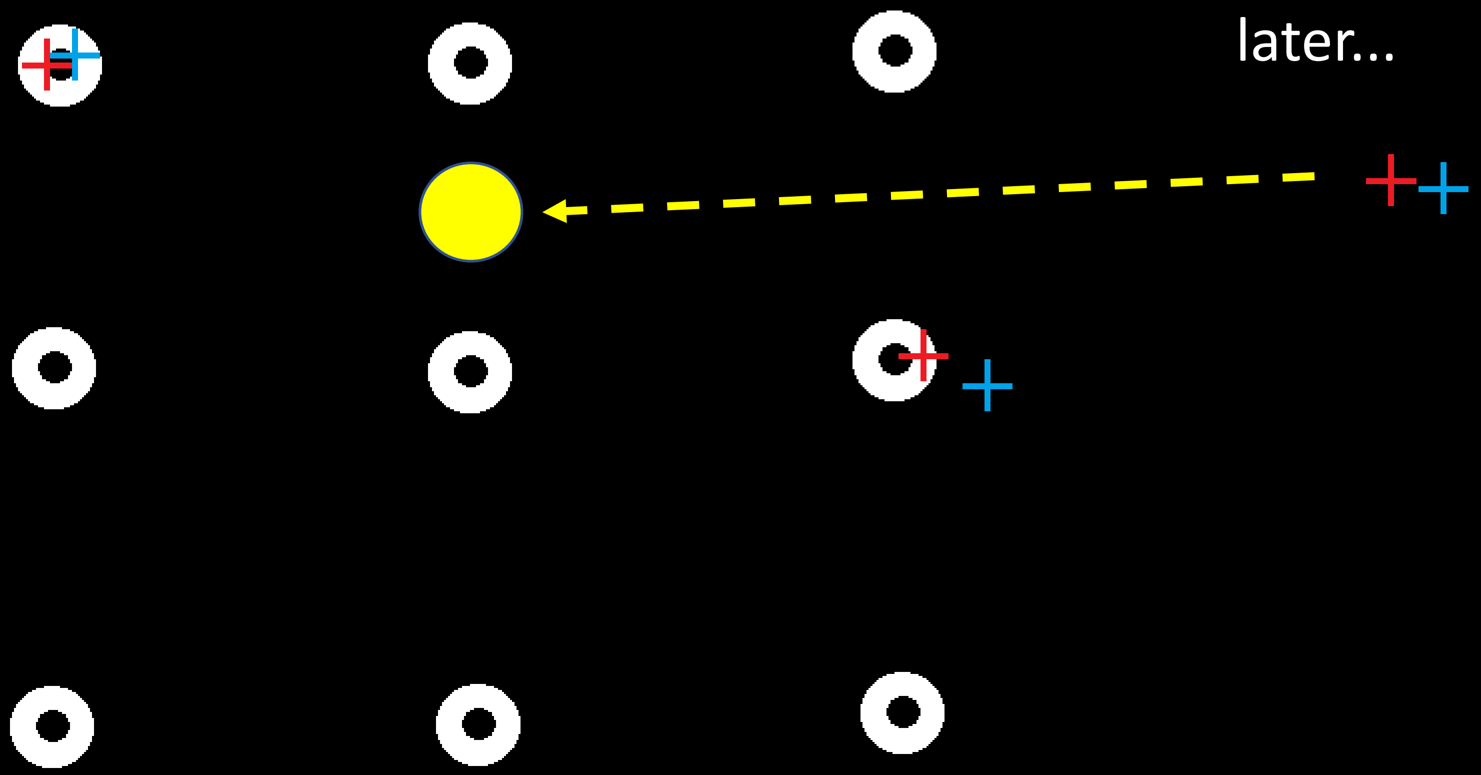
# Eye position: calibrate



Pupil location

Corneal reflection of (infrared) light sent from camera

# Eye position: calibrate

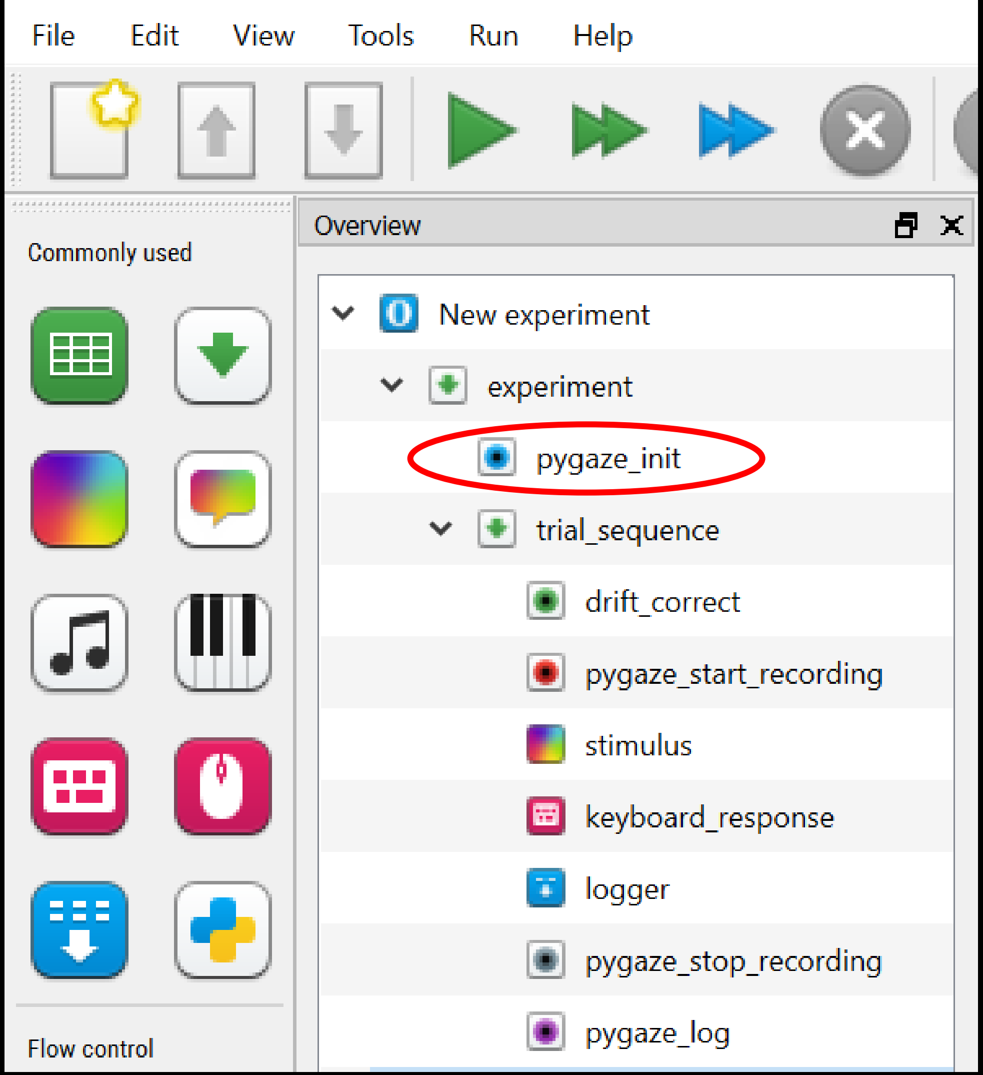


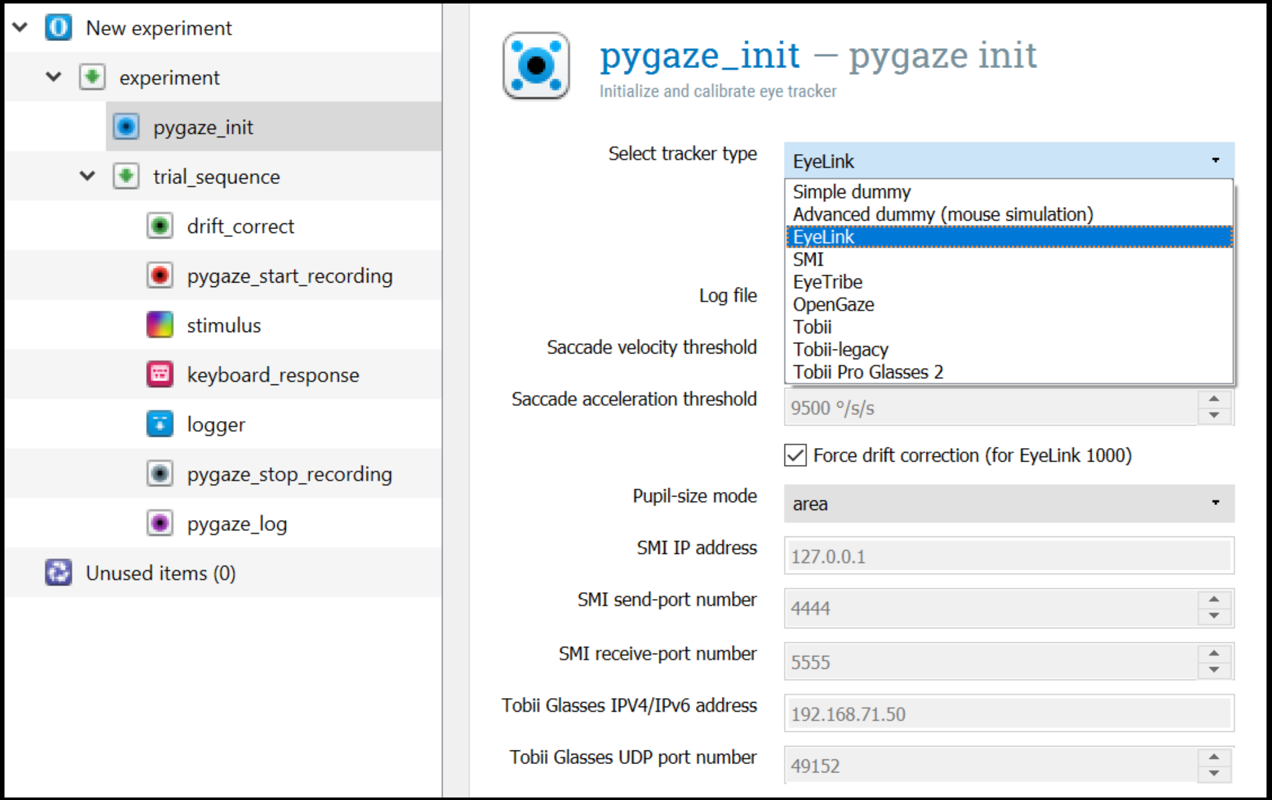
later...

Pupil location

Corneal reflection of (infrared) light sent from camera

# In OpenSesame...





**pygaze_init** — pygaze init

Initialize and calibrate eye tracker

Select tracker type: EyeLink

- Simple dummy
- Advanced dummy (mouse simulation)
- EyeLink
- SMI
- EyeTribe
- OpenGaze
- Tobii
- Tobii-legacy
- Tobii Pro Glasses 2

Log file

Saccade velocity threshold

Saccade acceleration threshold: 9500 °/s/s

☑ Force drift correction (for EyeLink 1000)

Pupil-size mode: area

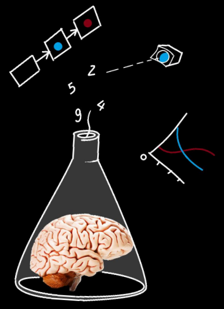
SMI IP address: 127.0.0.1

SMI send-port number: 4444

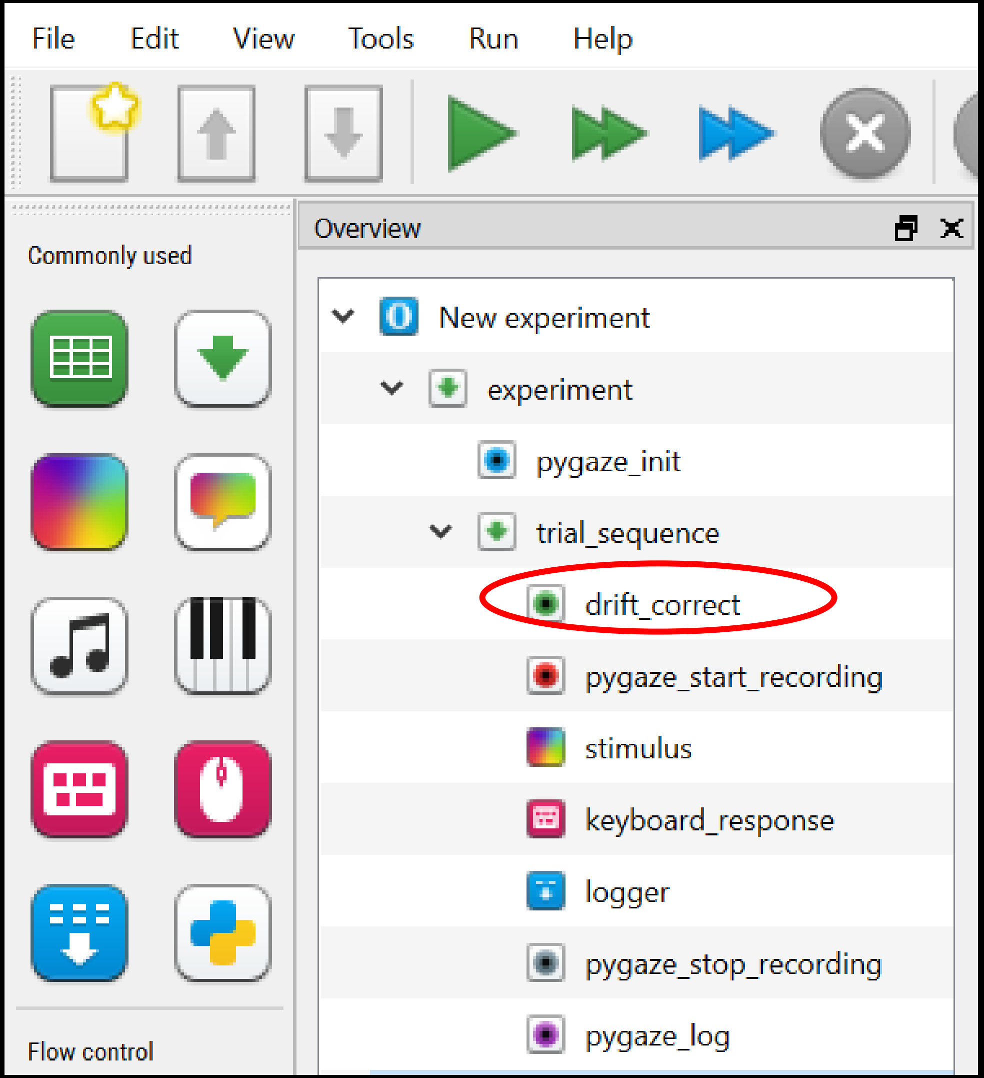
SMI receive-port number: 5555

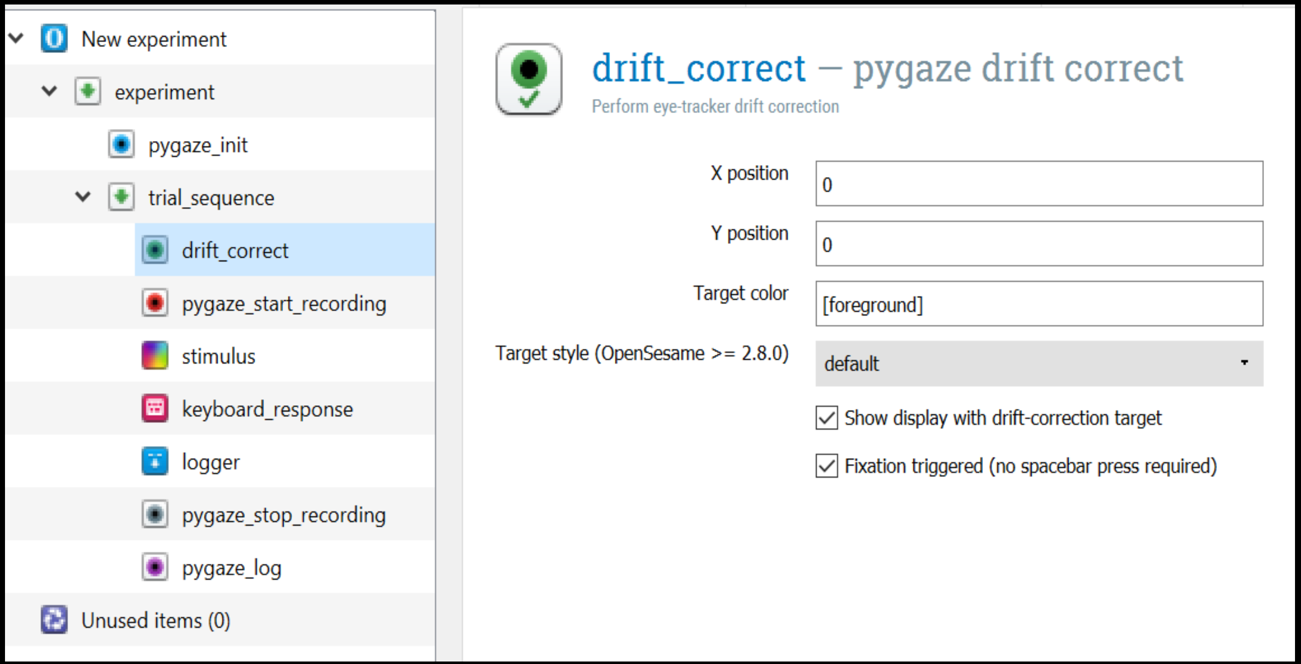
Tobii Glasses IPV4/IPv6 address: 192.168.71.50

Tobii Glasses UDP port number: 49152



# In OpenSesame...





drift_correct — pygaze drift correct

Perform eye-tracker drift correction

X position: 0

Y position: 0

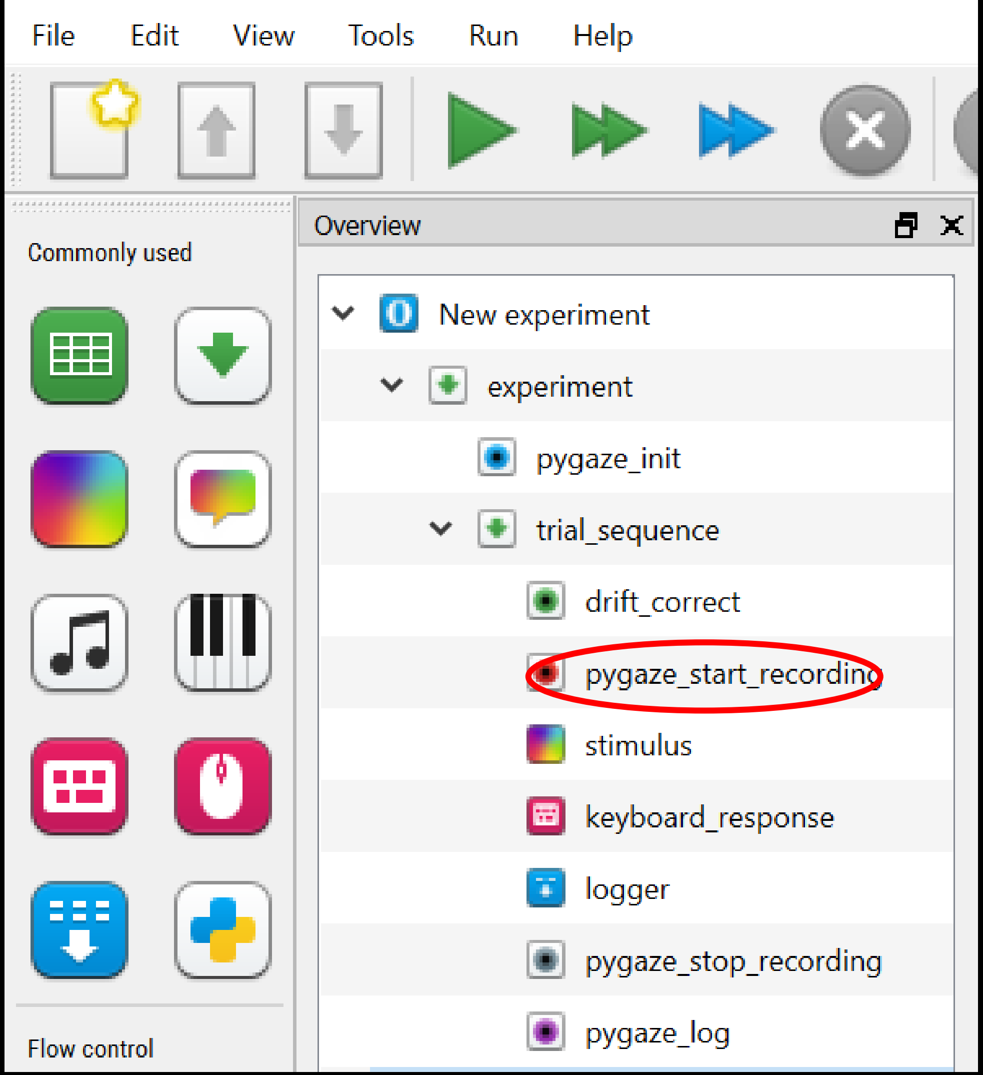
Target color: [foreground]

Target style (OpenSesame >= 2.8.0): default

☑ Show display with drift-correction target

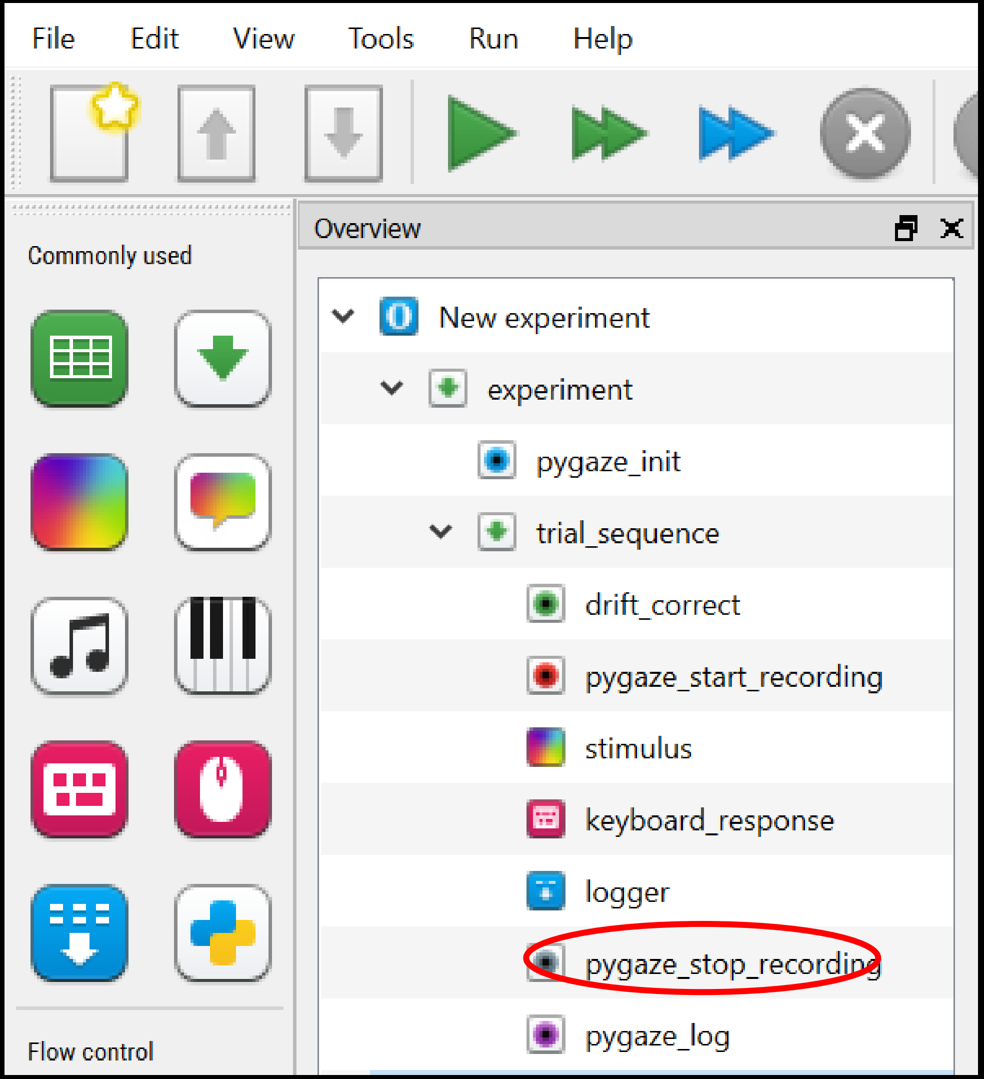
☑ Fixation triggered (no spacebar press required)

# In OpenSesame...



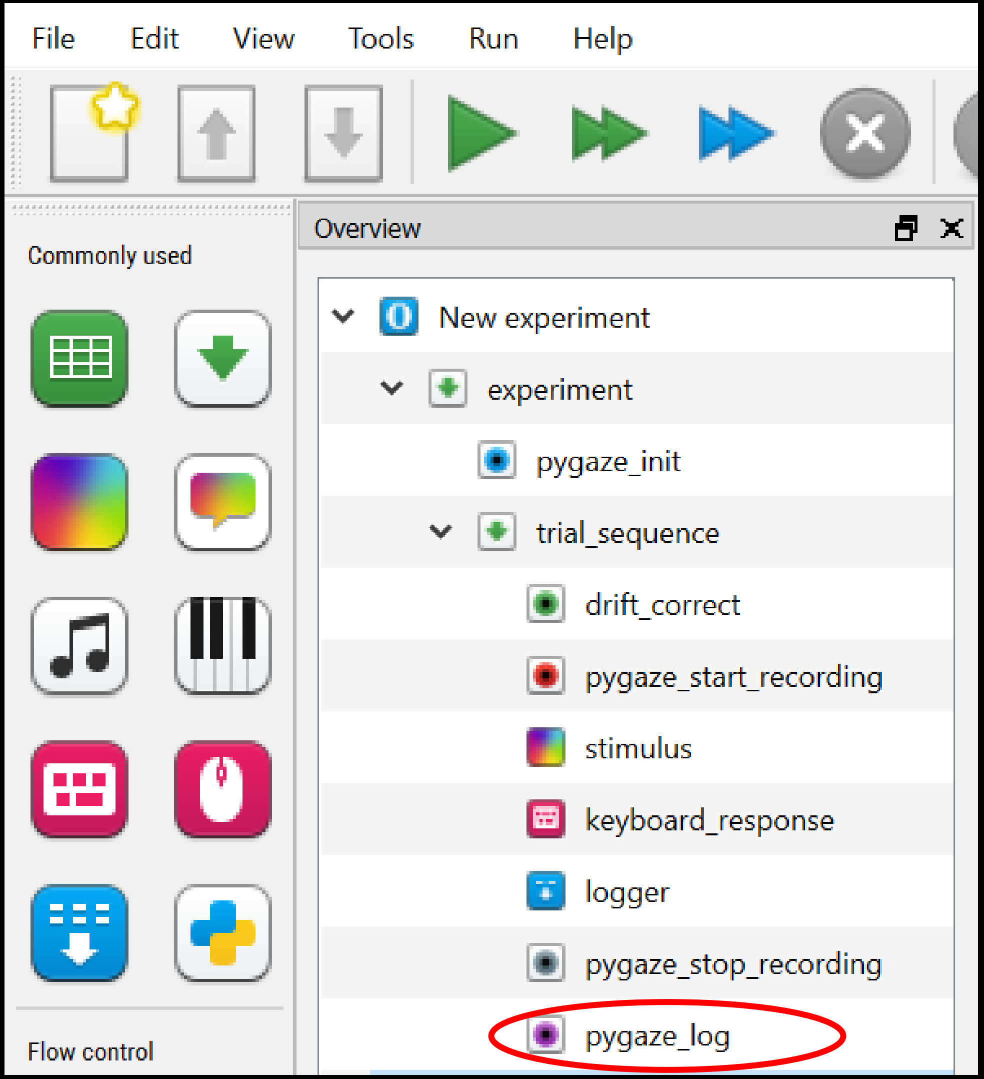
- New experiment
  - experiment
    - pygaze_init
    - trial_sequence
      - drift_correct
      - pygaze_start_recording
      - stimulus
      - keyboard_response
      - logger
      - pygaze_stop_recording
      - pygaze_log

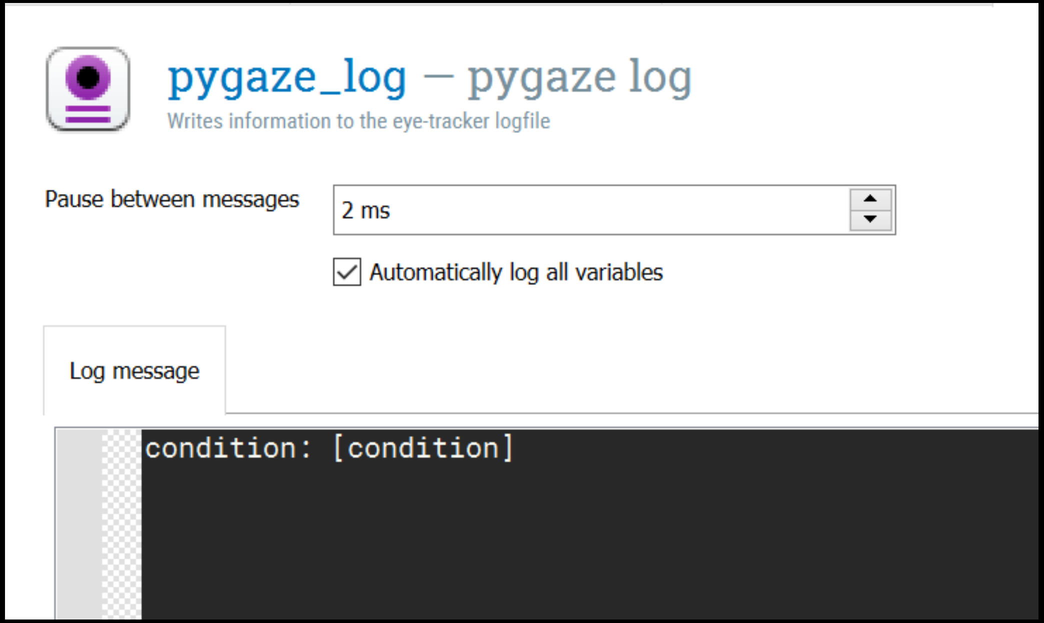
# In OpenSesame...



- New experiment
  - experiment
    - pygaze_init
    - trial_sequence
      - drift_correct
      - pygaze_start_recording
      - stimulus
      - keyboard_response
      - logger
      - pygaze_stop_recording
      - pygaze_log

# In OpenSesame…




pygaze_log — pygaze log

Writes information to the eye-tracker logfile

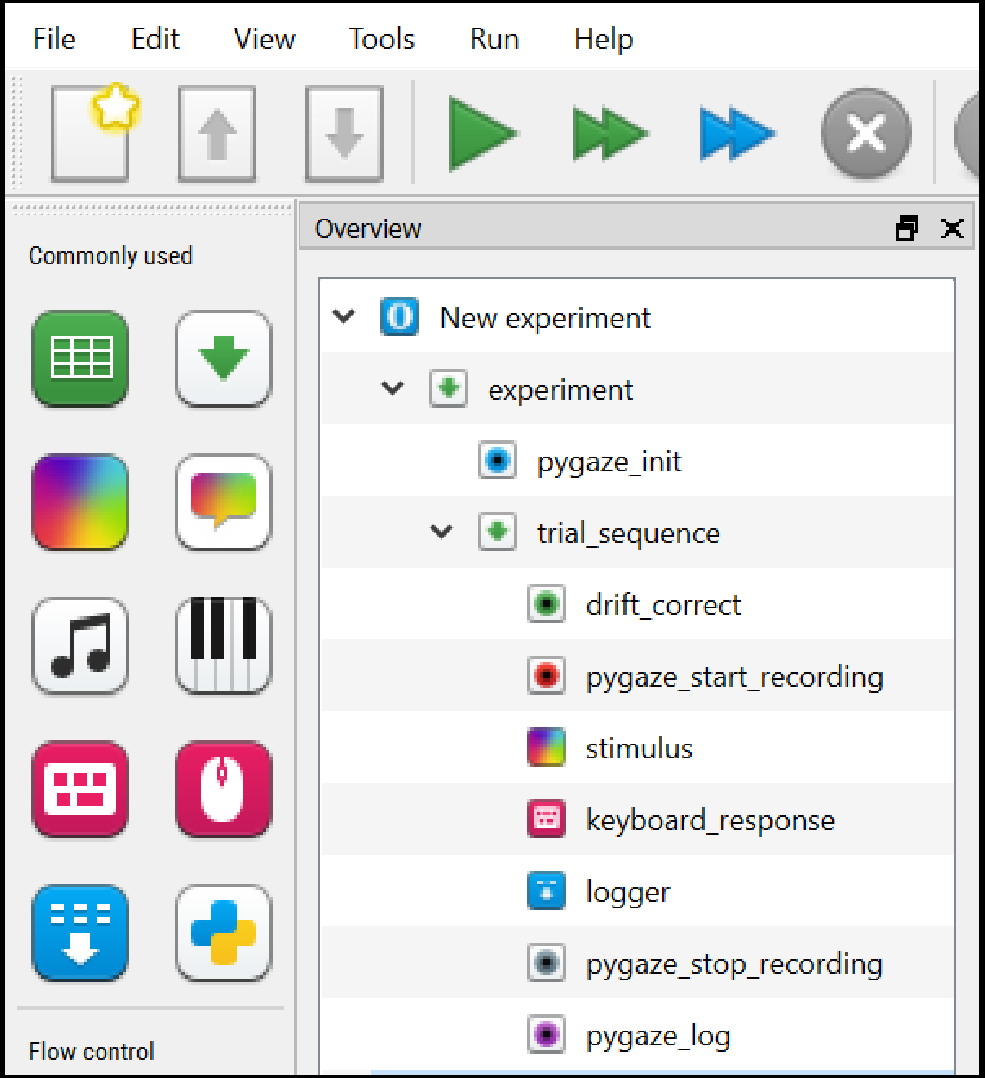
Pause between messages: 2 ms

☑ Automatically log all variables

Log message

```
condition: [condition]
```

# In OpenSesame…

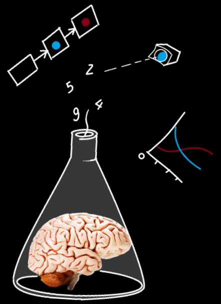


New experiment
- experiment
  - pygaze_init
  - trial_sequence
    - drift_correct
    - pygaze_start_recording
    - stimulus
    - keyboard_response
    - logger
    - pygaze_stop_recording
    - pygaze_log

Thus far:

Tracking the eye position (and pupil size) in a normal behavioral experiment

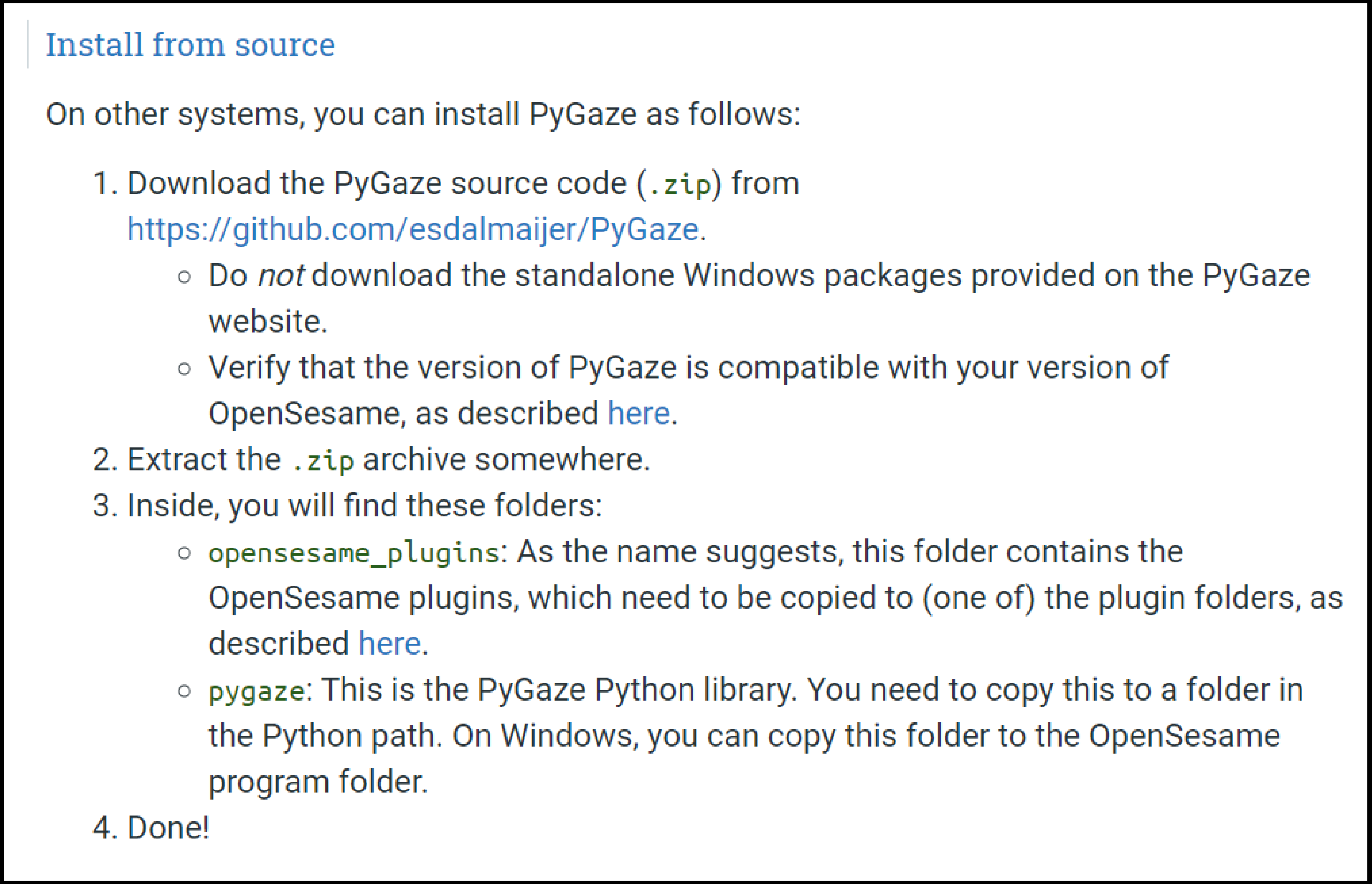
But what about gaze-contingent trickery?



# From today's module on Canvas, download *eyetracking.osexp*

Mac users:

may have to install PyGaze package manually



### Install from source

On other systems, you can install PyGaze as follows:

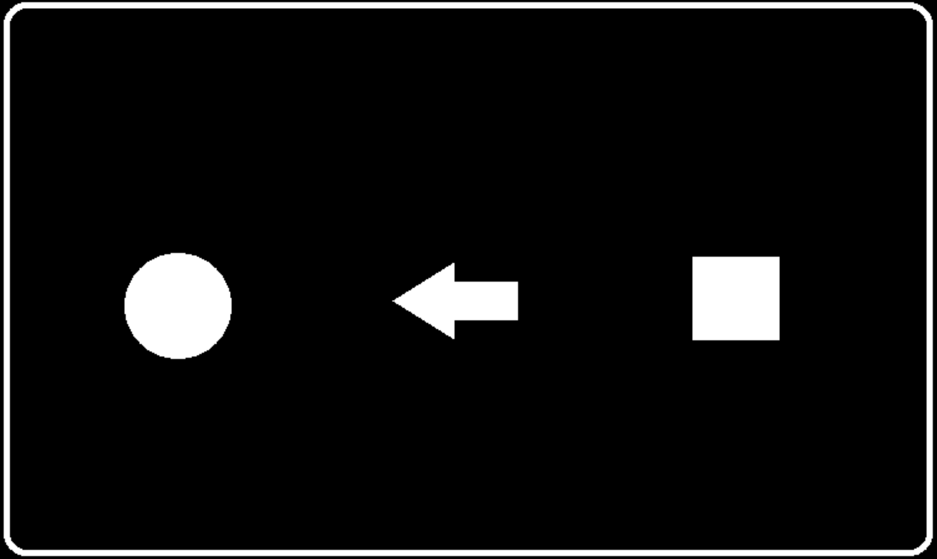
1. Download the PyGaze source code (`.zip`) from https://github.com/esdalmaijer/PyGaze.
    - Do *not* download the standalone Windows packages provided on the PyGaze website.
    - Verify that the version of PyGaze is compatible with your version of OpenSesame, as described here.
2. Extract the `.zip` archive somewhere.
3. Inside, you will find these folders:
    - `opensesame_plugins`: As the name suggests, this folder contains the OpenSesame plugins, which need to be copied to (one of) the plugin folders, as described here.
    - `pygaze`: This is the PyGaze Python library. You need to copy this to a folder in the Python path. On Windows, you can copy this folder to the OpenSesame program folder.
4. Done!

# A variant of the Posner cueing task

## Attention is biased by top-down cues

*e.g., it takes longer for you to note the square on the right, when the arrow points to the left.*

Task: indicate location of square (left / right)



***Potential confound:***

***Simon effect*** (Maybe arrow biased response rather than attention)

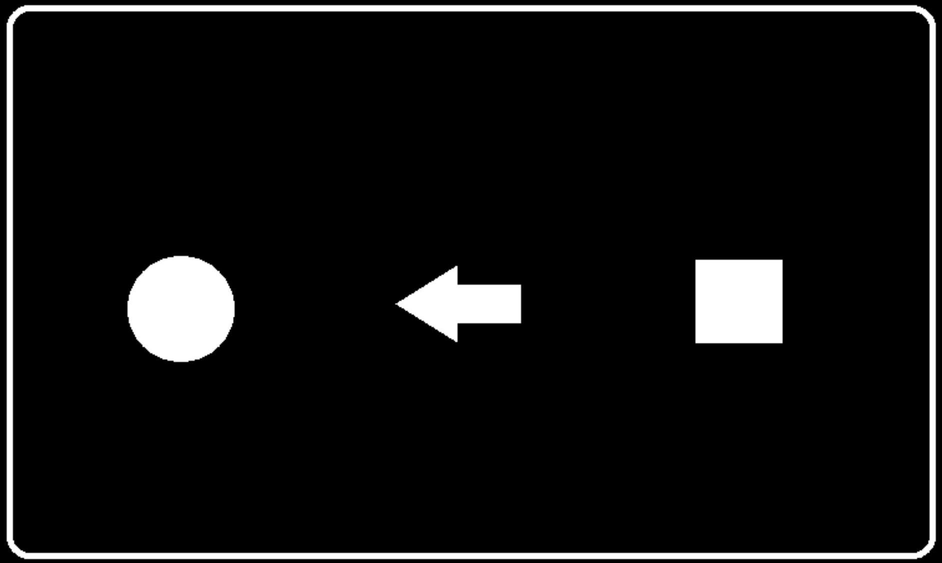
# A variant of the Posner cueing task

## Attention is biased by top-down cues

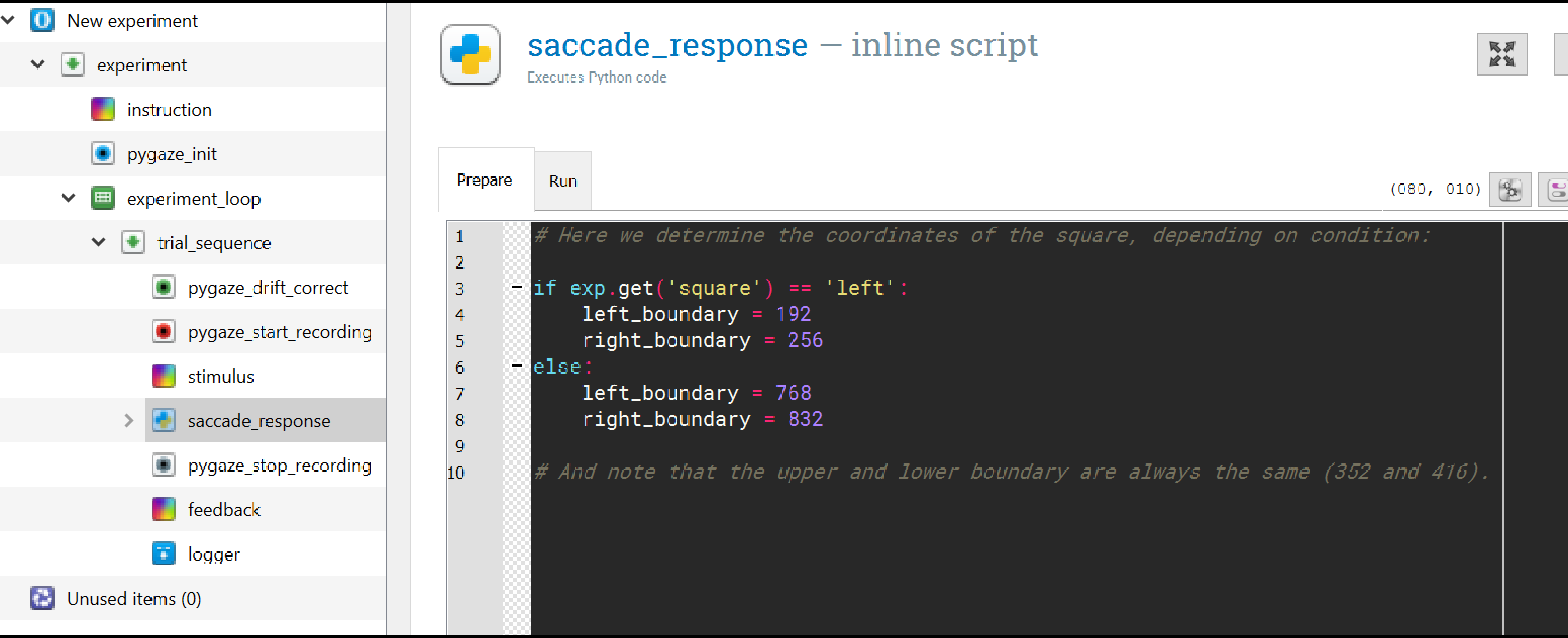
*e.g., it takes longer for you to note the square on the right, when the arrow points to the left.*

Task: *Move eyes to the square*

***Potential confound averted***

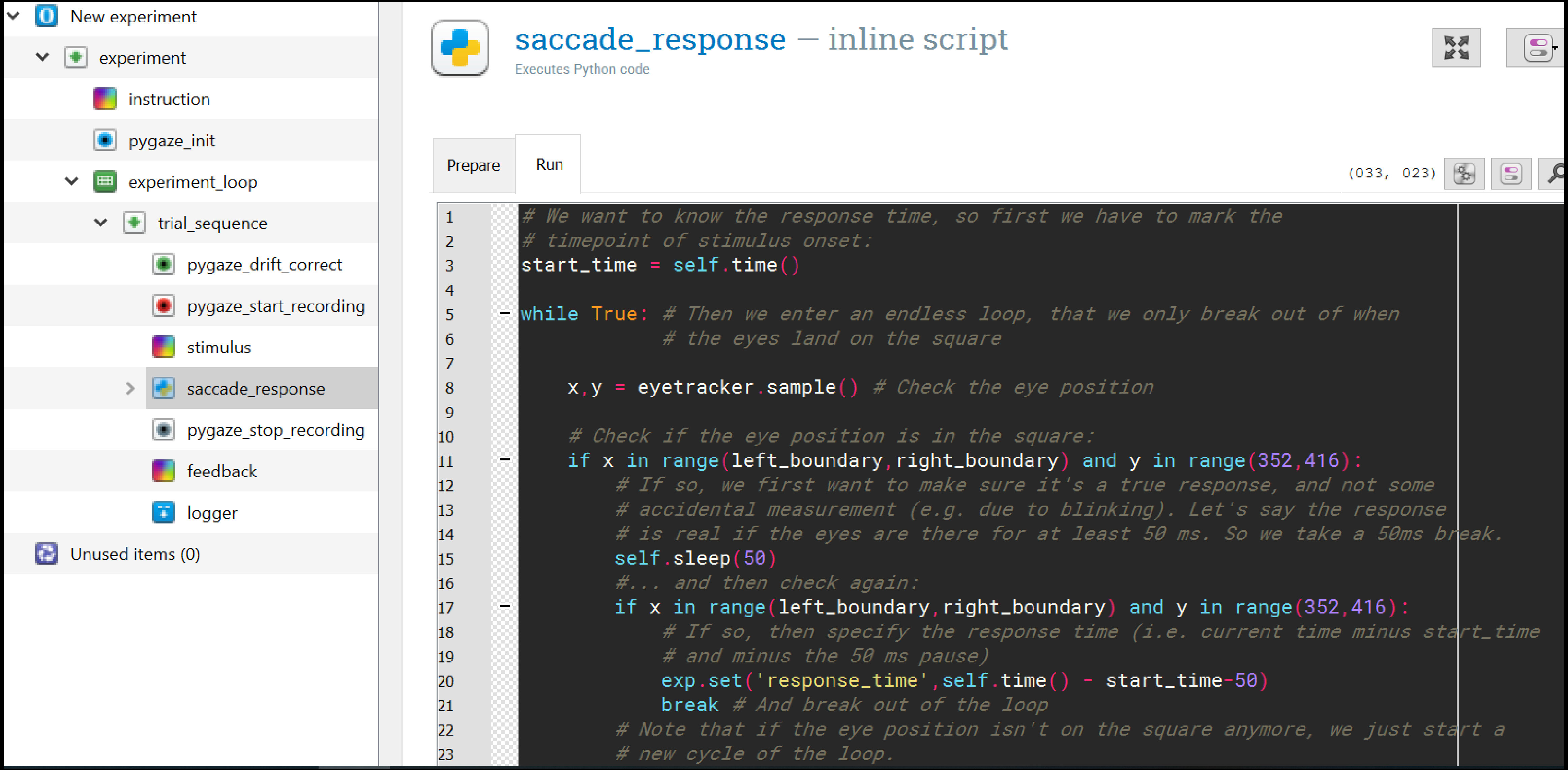


# A variant of the Posner cueing task



```python
# Here we determine the coordinates of the square, depending on condition:

if exp.get('square') == 'left':
    left_boundary = 192
    right_boundary = 256
else:
    left_boundary = 768
    right_boundary = 832

# And note that the upper and lower boundary are always the same (352 and 416).
```

# A variant of the Posner cueing task



```python
# We want to know the response time, so first we have to mark the
# timepoint of stimulus onset:
start_time = self.time()

while True: # Then we enter an endless loop, that we only break out of when
            # the eyes land on the square

    x,y = eyetracker.sample() # Check the eye position

    # Check if the eye position is in the square:
    if x in range(left_boundary,right_boundary) and y in range(352,416):
        # If so, we first want to make sure it's a true response, and not some
        # accidental measurement (e.g. due to blinking). Let's say the response
        # is real if the eyes are there for at least 50 ms. So we take a 50ms break.
        self.sleep(50)
        #... and then check again:
        if x in range(left_boundary,right_boundary) and y in range(352,416):
            # If so, then specify the response time (i.e. current time minus start_time
            # and minus the 50 ms pause)
            exp.set('response_time',self.time() - start_time-50)
            break # And break out of the loop
        # Note that if the eye position isn't on the square anymore, we just start a
        # new cycle of the loop.
```

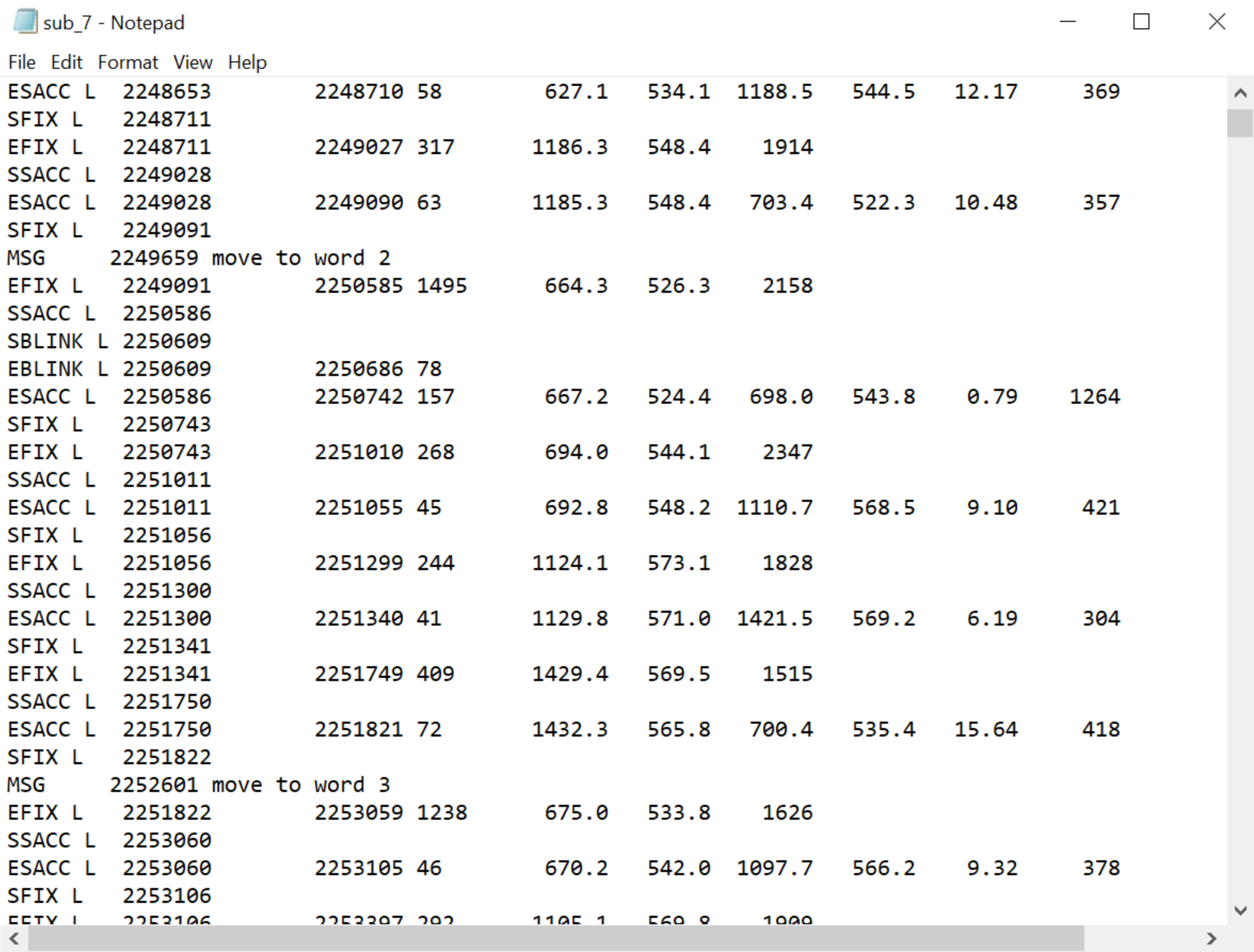
# Eye-tracker data

In our experiment we have a simple response_time variable…

*Do we need more?*

If possible, avoid having to dive into these files

→ define DV's *in* OpenSesame

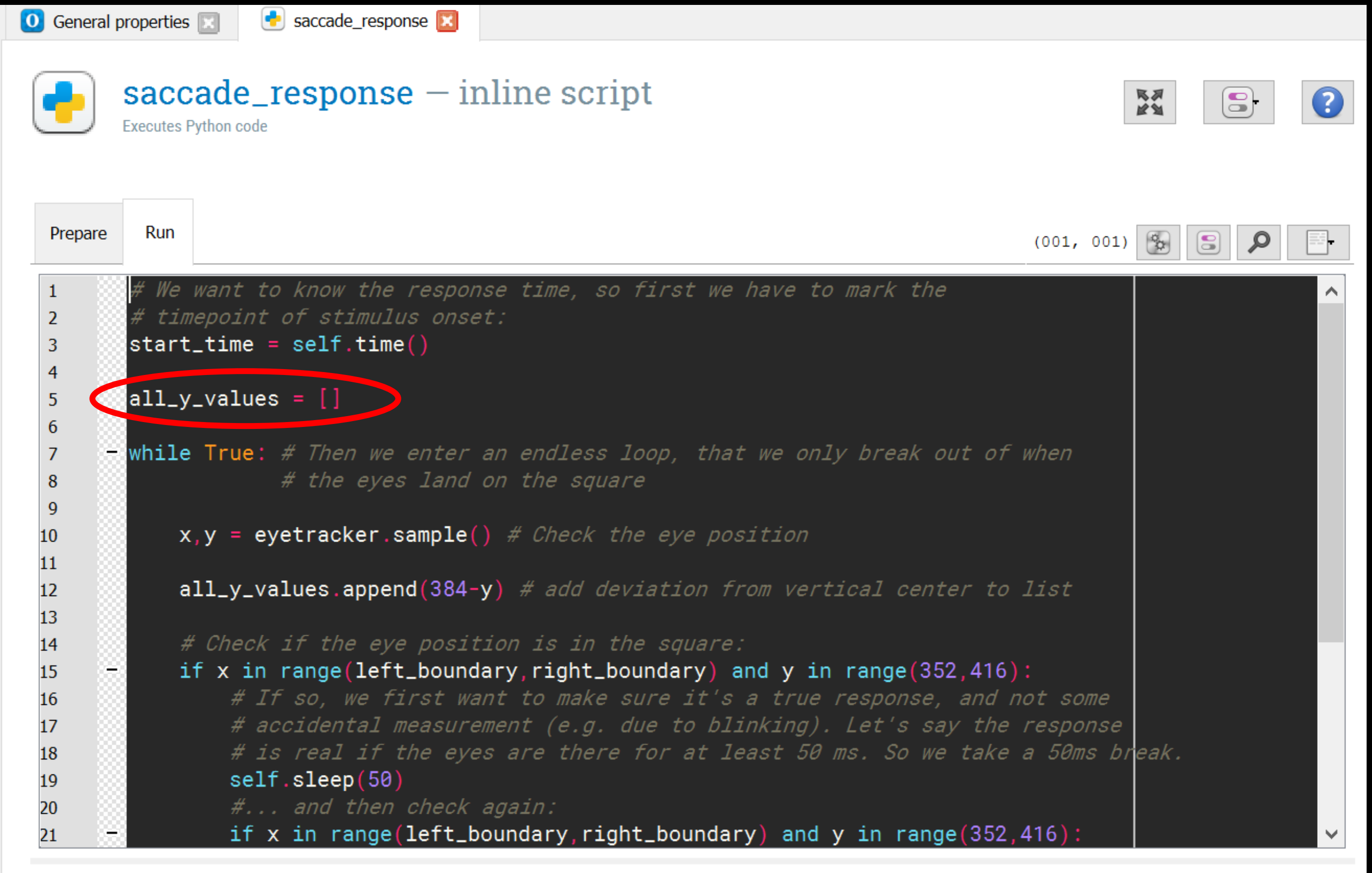


```
ESACC L   2248653        2248710 58        627.1    534.1  1188.5    544.5    12.17      369
SFIX L    2248711
EFIX L    2248711        2249027 317      1186.3    548.4    1914
SSACC L   2249028
ESACC L   2249028        2249090 63       1185.3    548.4   703.4    522.3    10.48      357
SFIX L    2249091
MSG      2249659 move to word 2
EFIX L    2249091        2250585 1495      664.3    526.3    2158
SSACC L   2250586
SBLINK L 2250609
EBLINK L 2250609         2250686 78
ESACC L   2250586        2250742 157       667.2    524.4   698.0    543.8     0.79     1264
SFIX L    2250743
EFIX L    2250743        2251010 268       694.0    544.1    2347
SSACC L   2251011
ESACC L   2251011        2251055 45        692.8    548.2  1110.7    568.5     9.10      421
SFIX L    2251056
EFIX L    2251056        2251299 244      1124.1    573.1    1828
SSACC L   2251300
ESACC L   2251300        2251340 41       1129.8    571.0  1421.5    569.2     6.19      304
SFIX L    2251341
EFIX L    2251341        2251749 409      1429.4    569.5    1515
SSACC L   2251750
ESACC L   2251750        2251821 72       1432.3    565.8   700.4    535.4    15.64      418
SFIX L    2251822
MSG      2252601 move to word 3
EFIX L    2251822        2253059 1238      675.0    533.8    1626
SSACC L   2253060
ESACC L   2253060        2253105 46        670.2    542.0  1097.7    566.2     9.32      378
SFIX L    2253106
EFIX L    2253106        2253397 292      1105.1    569.8    1909
```

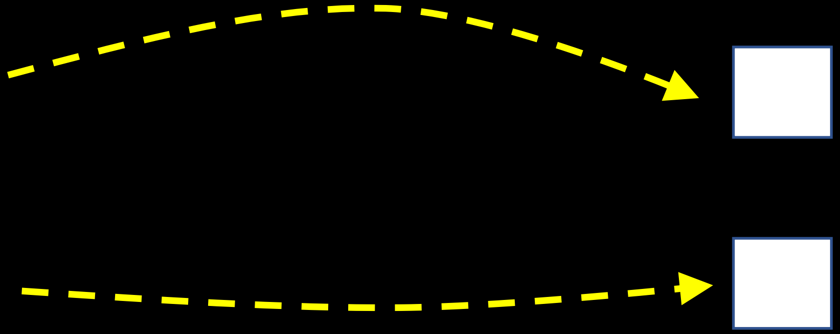
# Eye-tracker data

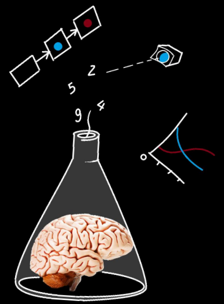
→ define DV's *in* OpenSesame

*Example: saccadic curvature*



```
# We want to know the response time, so first we have to mark the
# timepoint of stimulus onset:
start_time = self.time()

all_y_values = []

while True: # Then we enter an endless loop, that we only break out of when
            # the eyes land on the square

    x,y = eyetracker.sample() # Check the eye position

    all_y_values.append(384-y) # add deviation from vertical center to list

    # Check if the eye position is in the square:
    if x in range(left_boundary,right_boundary) and y in range(352,416):
        # If so, we first want to make sure it's a true response, and not some
        # accidental measurement (e.g. due to blinking). Let's say the response
        # is real if the eyes are there for at least 50 ms. So we take a 50ms break.
        self.sleep(50)
        #... and then check again:
        if x in range(left_boundary,right_boundary) and y in range(352,416):
```

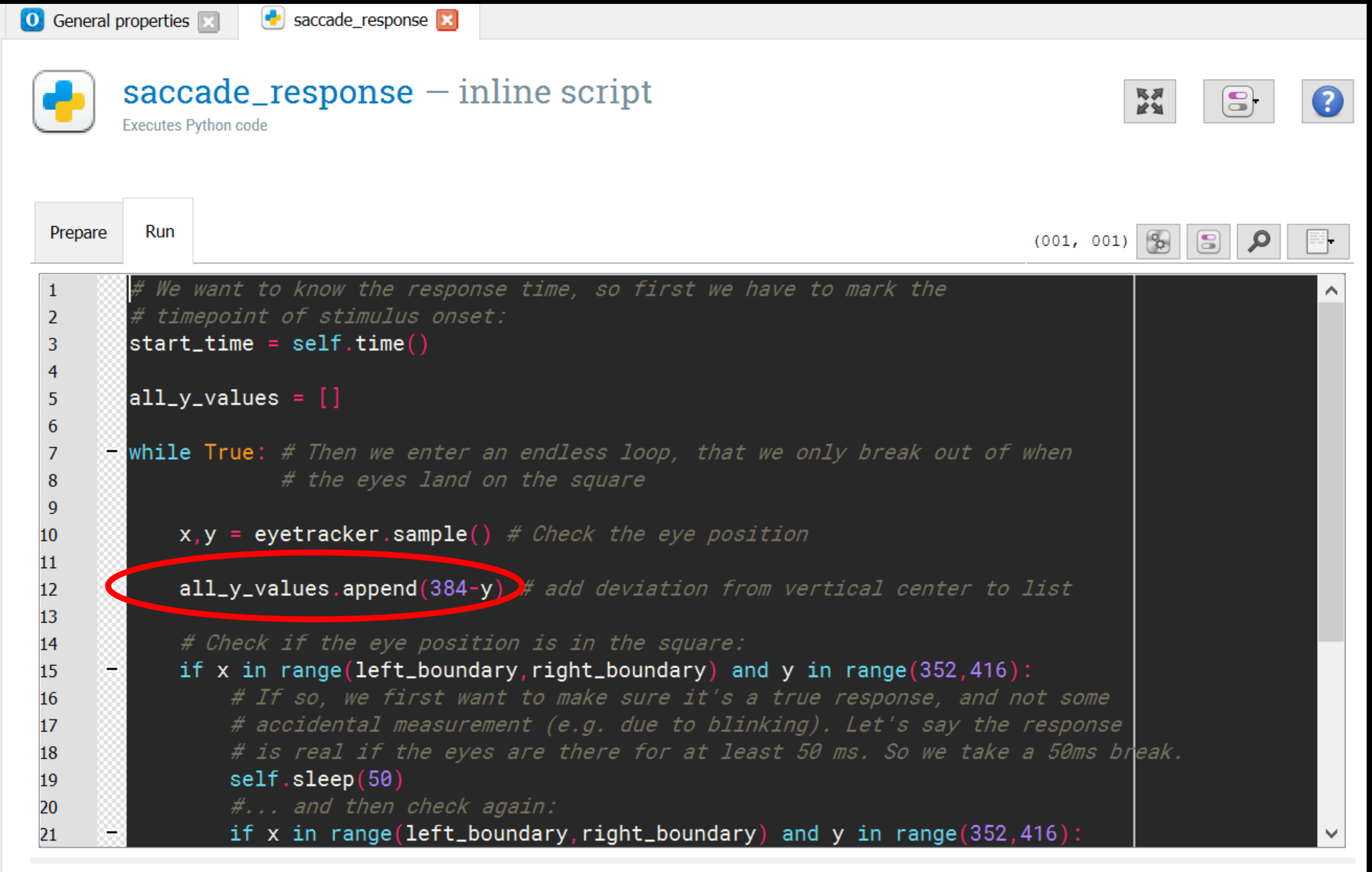




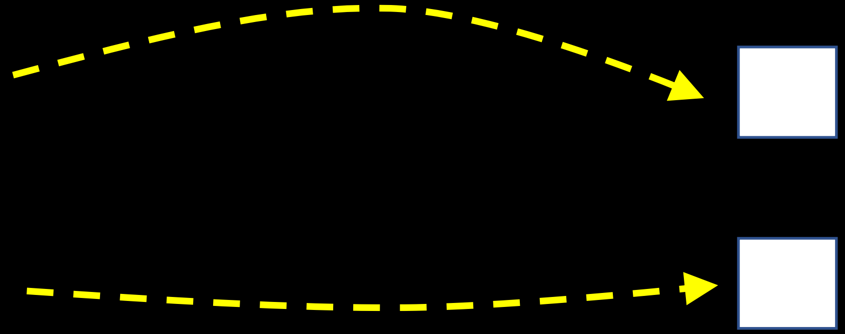
# Eye-tracker data

## → define DV's *in* OpenSesame

## *Example: saccadic curvature*



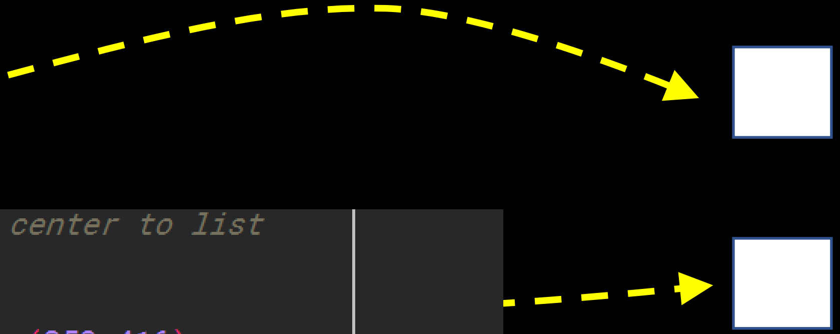
```
# We want to know the response time, so first we have to mark the
# timepoint of stimulus onset:
start_time = self.time()

all_y_values = []

while True: # Then we enter an endless loop, that we only break out of when
            # the eyes land on the square

    x,y = eyetracker.sample() # Check the eye position

    all_y_values.append(384-y)# add deviation from vertical center to list

    # Check if the eye position is in the square:
    if x in range(left_boundary,right_boundary) and y in range(352,416):
        # If so, we first want to make sure it's a true response, and not some
        # accidental measurement (e.g. due to blinking). Let's say the response
        # is real if the eyes are there for at least 50 ms. So we take a 50ms break.
        self.sleep(50)
        #... and then check again:
        if x in range(left_boundary,right_boundary) and y in range(352,416):
```

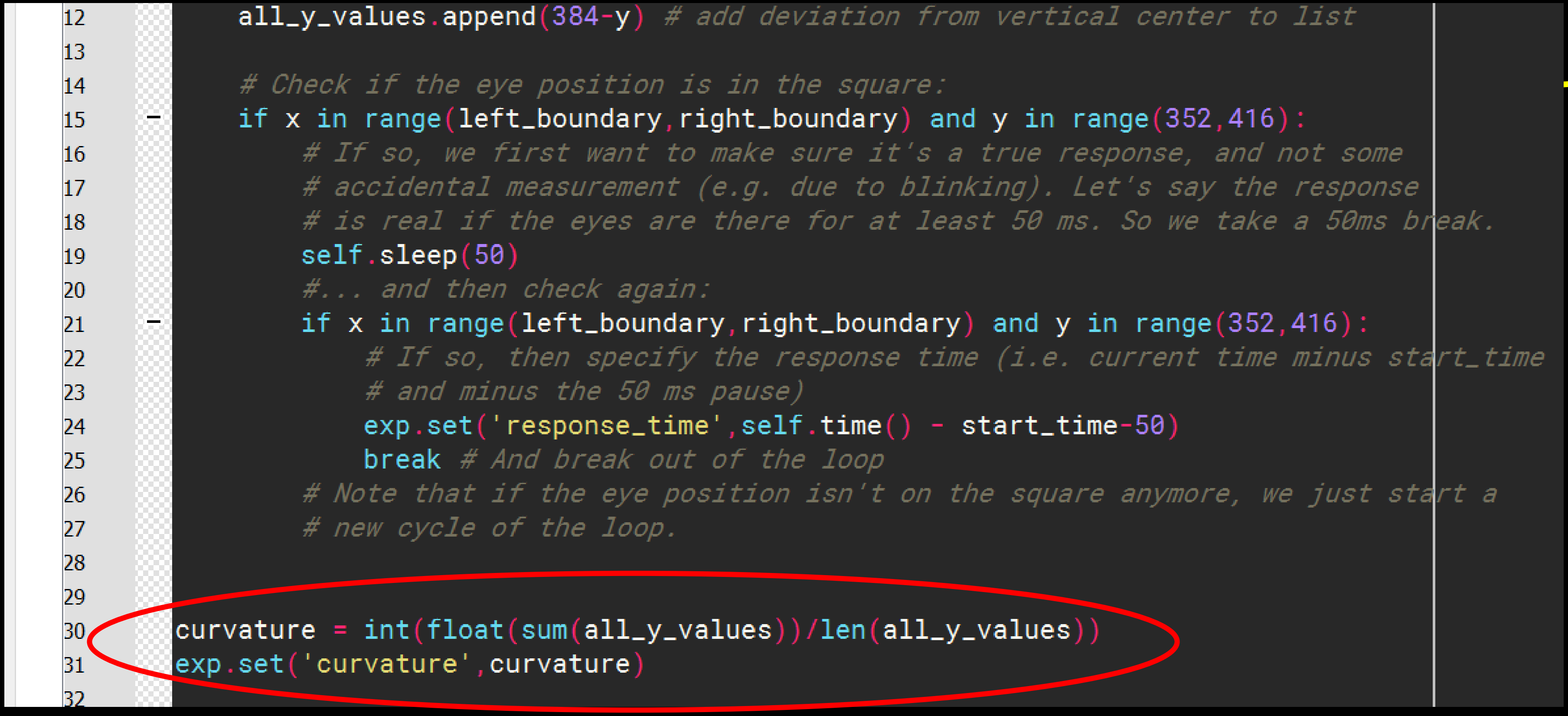


# Eye-tracker data

→ define DV's *in* OpenSesame

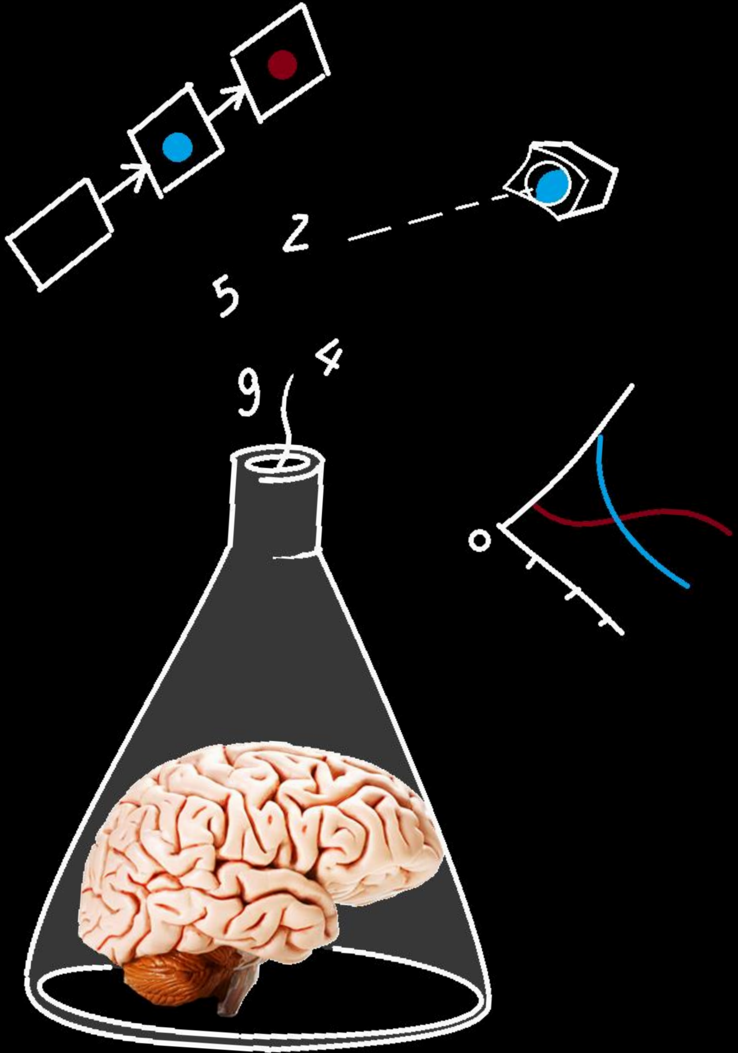
*Example: saccadic curvature*





```
    all_y_values.append(384-y) # add deviation from vertical center to list

    # Check if the eye position is in the square:
    if x in range(left_boundary,right_boundary) and y in range(352,416):
        # If so, we first want to make sure it's a true response, and not some
        # accidental measurement (e.g. due to blinking). Let's say the response
        # is real if the eyes are there for at least 50 ms. So we take a 50ms break.
        self.sleep(50)
        #... and then check again:
        if x in range(left_boundary,right_boundary) and y in range(352,416):
            # If so, then specify the response time (i.e. current time minus start_time
            # and minus the 50 ms pause)
            exp.set('response_time',self.time() - start_time-50)
            break # And break out of the loop
        # Note that if the eye position isn't on the square anymore, we just start a
        # new cycle of the loop.


curvature = int(float(sum(all_y_values))/len(all_y_values))
exp.set('curvature',curvature)
```

# Pupillometry practical



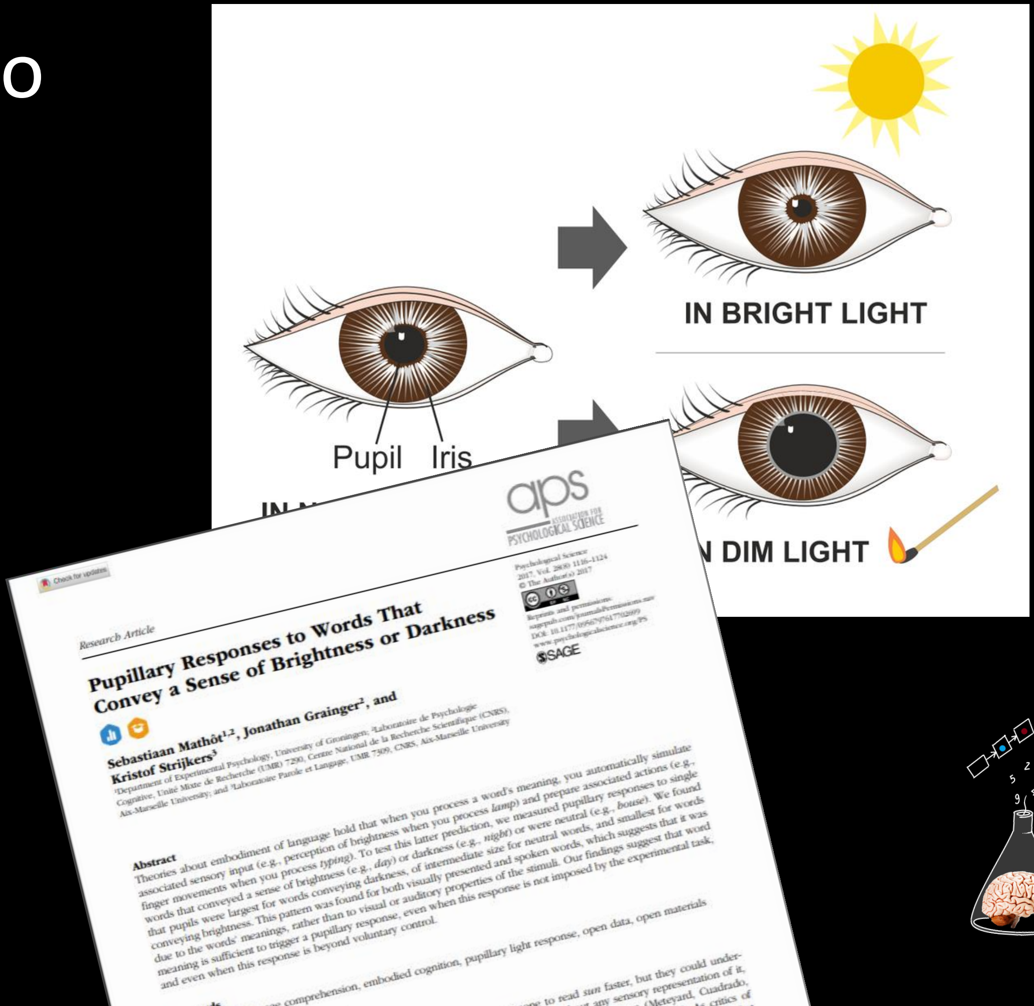
# Pupillometry practical

*Assignment due: Sunday 1st 23:59*

Pupillary light response: not just to our direct visual environment

Also to memorized brightness

(Mathôt et al., 2017: pupil response to semantic brightness of words)



IN BRIGHT LIGHT
Pupil
Iris
IN N
N DIM LIGHT

Check for updates

Research Article

Psychological Science
2017, Vol. 28(8) 1116–1124
© The Author(s) 2017
Reprints and permissions:
sagepub.com/journalsPermissions.nav
DOI: 10.1177/0956797617702699
www.psychologicalscience.org/PS
SAGE

**Pupillary Responses to Words That Convey a Sense of Brightness or Darkness**

**Sebastiaan Mathôt[1,2], Jonathan Grainger[2], and Kristof Strijkers[3]**

[1]Department of Experimental Psychology, University of Groningen; [2]Laboratoire de Psychologie Cognitive, Unité Mixte de Recherche (UMR) 7290, Centre National de la Recherche Scientifique (CNRS), Aix-Marseille University; and [3]Laboratoire Parole et Langage, UMR 7309, CNRS, Aix-Marseille University

**Abstract**

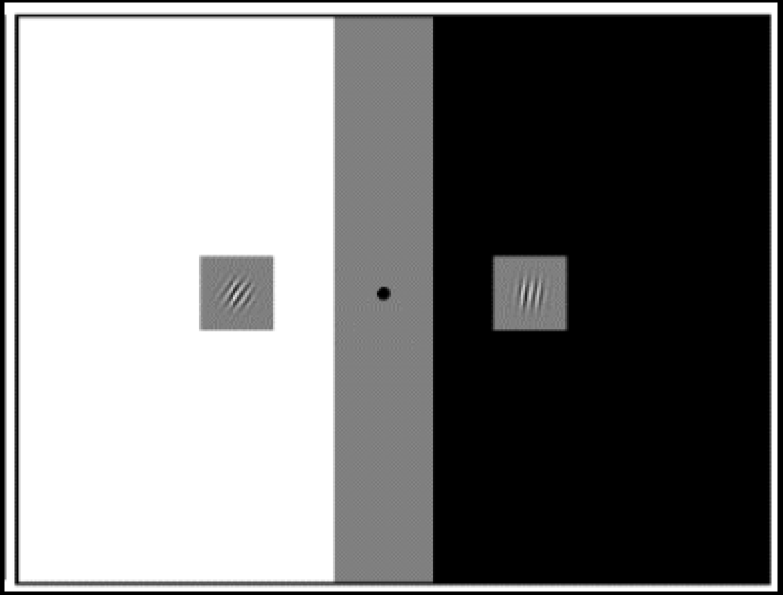
Theories about embodiment of language hold that when you process a word's meaning, you automatically simulate associated sensory input (e.g., perception of brightness when you process *lamp*) and prepare associated actions (e.g., finger movements when you process *typing*). To test this latter prediction, we measured pupillary responses to single words that conveyed a sense of brightness (e.g., *day*) or darkness (e.g., *night*) or were neutral (e.g., *house*). We found that pupils were largest for words conveying darkness, of intermediate size for neutral words, and smallest for words conveying brightness. This pattern was found for both visually presented and spoken words, which suggests that it was due to the words' meanings, rather than to visual or auditory properties of the stimuli. Our findings suggest that word meaning is sufficient to trigger a pupillary response, even when this response is not imposed by the experimental task, and even when this response is beyond voluntary control.

ge comprehension, embodied cognition, pupillary light response, open data, open materials

ne to read *sun* faster, but they could under- any sensory representation of it, (Meteyard, Cuadrado, critics of

# Pupillometry practical

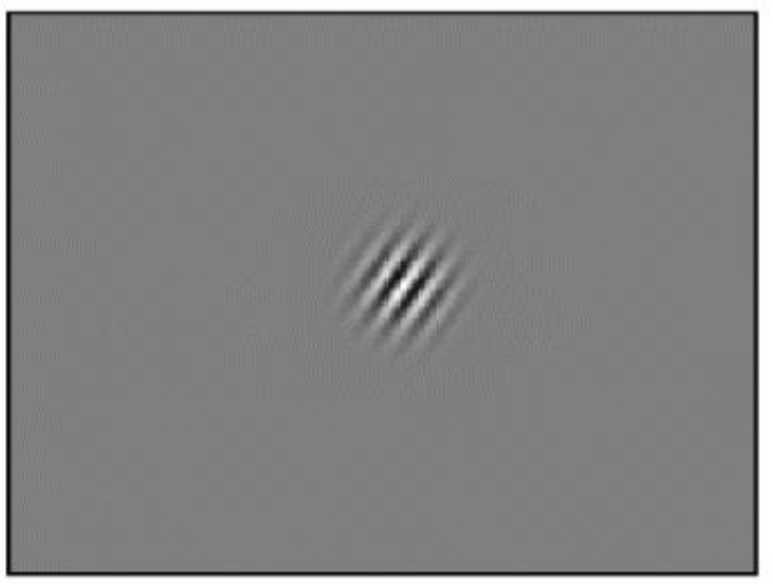
*Assignment due: Sunday 1st 23:59*



# Pupillometry practical

*Assignment due: Sunday 1st 23:59*

“have you seen this
orientation?”



H: pupil responds to brightness
of memorized stimulus location

# Pupillometry practical

*Assignment due: Sunday 1st 23:59*

*From today's module in Canvas, download*

pupillometry practical.pdf

Assignment consists of 2 parts; Part 2 is about analyzing data (*Thursday!*)