

# Prepare your work folder

**# Update the Materials folder to the latest version of our GitHub**

```
cd your-path/Programming/Materials  
git pull
```

**# Create a Week-5 folder in Assignments**

```
cd your-path/Programming/Assignments  
mkdir Week-5  
cd Week-5
```

**# Copy the contents of Materials/Week-5 into Assignments/Week-5**

```
cp -R ../../Materials/Week-5/* .
```

# Experiment: Keyboard input and data collection

**Programming Psychology Experiments (CORE-1)**

Barbu Revencu & Maxime Cauté

Session 5 | 8 October 2025

# The plan for today

1. **Assignment** discussion
2. **Get participant (keyboard) input**
3. **Record participant data**

# Assignment 3 Discussion

# drawing\_functions.py

```
FPS = 60 # frames per second
MSPF = 1000 / FPS # milliseconds per frame (more robust than using 16.67)

def to_frames(t):
    return t / MSPF

def to_time(num_frames):
    return num_frames * MSPF

def load(stims):
    for stim in stims:
        stim.preload()
```

# drawing\_functions.py

```
def timed_draw(stims):  
    t0 = exp.clock.time # Initial time  
  
    exp.screen.clear()  
    for stim in stims:  
        stim.present(clear=False, update=False)  
    exp.screen.update()  
  
    elapsed = exp.clock.time - t0 # Time after drawing  
    return elapsed  
  
def present_for(stims, num_frames):  
    if num_frames == 0: # If num_frames = 0 → No need to present anything  
        return  
    dt = timed_draw(stims) # Get time needed for correction  
    if dt > 0:  
        t = to_time(num_frames) # Convert frames to time  
        exp.clock.wait(t - dt) # Adjust waiting time by dt
```

# ternus.py: Stimuli

```
""" Stimuli """
RADIUS = 50; DISTANCE = RADIUS * 3; SPREAD = RADIUS * 9

def make_circles(radius=RADIUS):
    positions = range(-SPREAD // 2, SPREAD // 2, DISTANCE) # x-positions: [-225, -75, 75]
    circles = [stimuli.Circle(radius=radius, position=(x_pos, 0)) for x_pos in positions]
    return circles

def add_tags(circles, tag_radius):
    tag_colors = [C_YELLOW, C_RED, C_BLUE]
    tag_circles = [stimuli.Circle(radius=tag_radius, colour=col) for col in tag_colors]
    for circle, tag in zip(circles, tag_circles):
        tag.plot(circle)

""" Inside your run_trial function """
circles = make_circles()
if tags: add_tags(circles, tag_radius=RADIUS // 5)
load(circles)
```

# ternus . py: Animation

```
""" Display loop """  
while True:  
    for dx in (extent, -extent):  
        present_for(circles, num_frames=12) # 200 ms  
        present_for([], num_frames=ISI)  
        circles[0].move((dx, 0))  
  
    if exp.keyboard.check(K_SPACE):  
        break
```



# Comment

The functions you wrote in `drawing_functions.py` were **not just an exercise**

Rather, you should **reuse** them to make your life easier whenever useful

One option would have been of course to copy-paste them on top of your `ternus` script, but this is tedious

# Import your helper functions

```
# IN DRAWING_FUNCTIONS.PY
def timed_draw(exp, stims):
    ...
    exp.screen.update()
    ...

def present_for(exp, stims, num_frames):
    ...
    dt = timed_draw(exp, stims)
    if dt > 0:
        t = to_time(num_frames)
        exp.clock.wait(t - dt)

# IN TERNUS.PY
from drawing_functions import *
...
present_for(exp, circles, num_frames=12)
```

# ternus . py: Trial

```
def run_trial(circle_frames=12, ISI=0, tags=False):  
  
    # Create circles  
    circles = make_circles()  
    if tags: add_tags(circles, tag_radius=RADIUS // 5)  
    load(circles)  
  
    while True:  
        for dx in (extent, -extent):  
            present_for(exp, circles, num_frames=circle_frames)  
            present_for(exp, [], num_frames=ISI)  
            circles[0].move((dx, 0))  
  
        if exp.keyboard.check(K_SPACE):  
            break
```

# ternus .py: Full

```
exp = design.Experiment("Ternus", background_colour=C_WHITE, foreground_colour=C_BLACK)

control.initialize(exp)
control.start(subject_id=1)

trials = [{'ISI': 0}, {'ISI': 18}, {'ISI': 18, 'tags': True}]

for trial_params in trials:
    run_trial(**trial_params)

control.end()
```

# Keyboard input

# `exp.keyboard.wait()`

We have used this function from the beginning whenever we wanted to pause the script and wait for *any* key press

Conveniently, this function gives us everything we need to start doing things based on specific inputs

# `exp.keyboard.wait()`

First, `keyboard.wait` is customizable:

1. It can take one or more keys as input, in which case it only checks for these *specific* key presses: `exp.keyboard.wait(keys=[...])`

2. You can add a timeout:

```
exp.keyboard.wait(duration=1000)
```

Second, the function **returns** the **key pressed** and the **time** until the press:

```
key, rt = exp.keyboard.wait(keys=[...])
```

# Specifying keys

In experiment, keys are represented as integers (ASCII code)

Your scripts therefore need to have access to the mapping between each key (e.g., 'a')

You can obtain it one of two ways:

1. Use python's built-in **ord** function: `ord('a') = 97`
2. Import from **experiment.misc.constants**: `K_a`

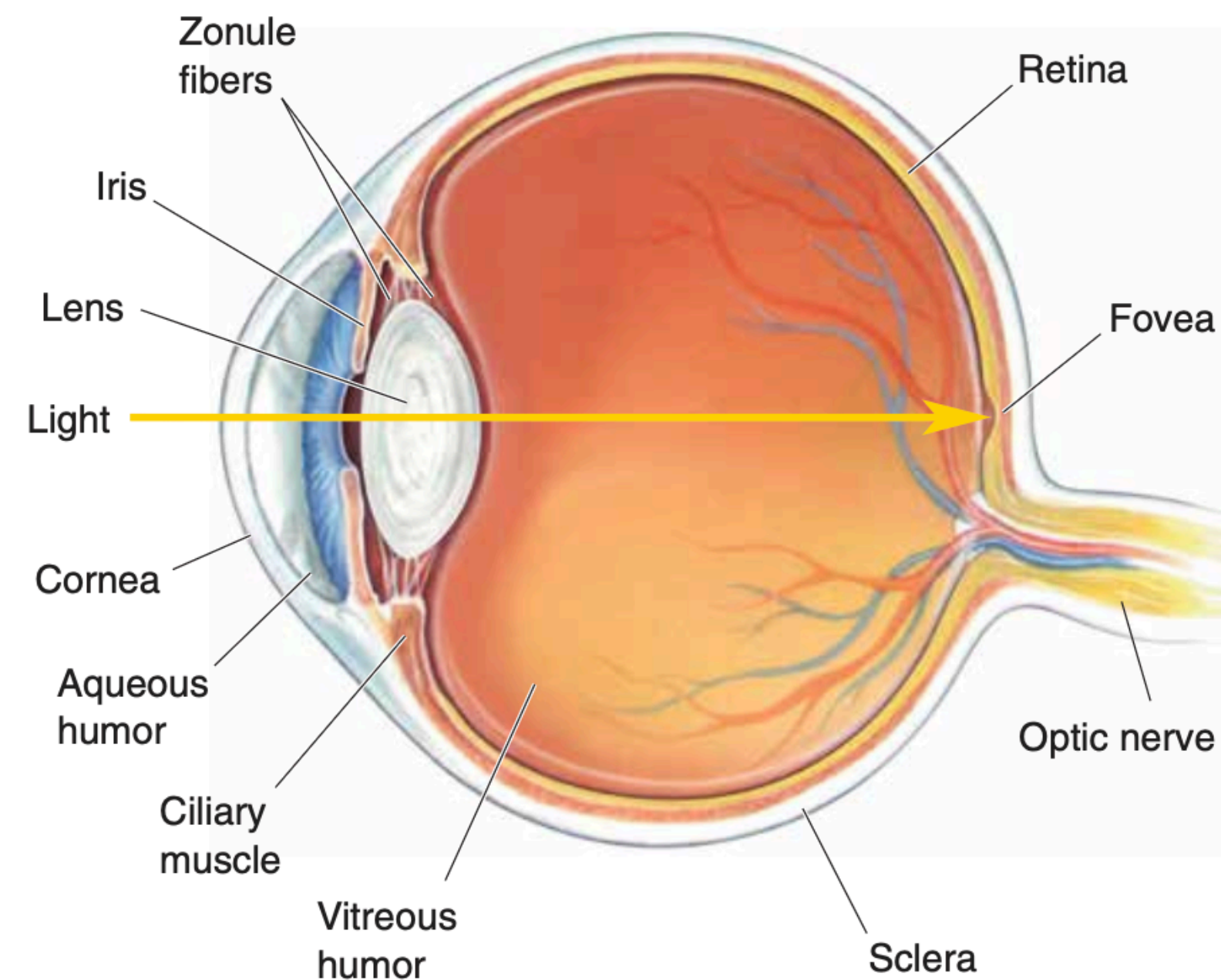


# Keyboard arrows

There is no single ASCII code for keyboard arrow keys, so it's better to import them from **experiment.misc.constants**: `K_DOWN`, `K_UP`, `K_LEFT`, `K_RIGHT`

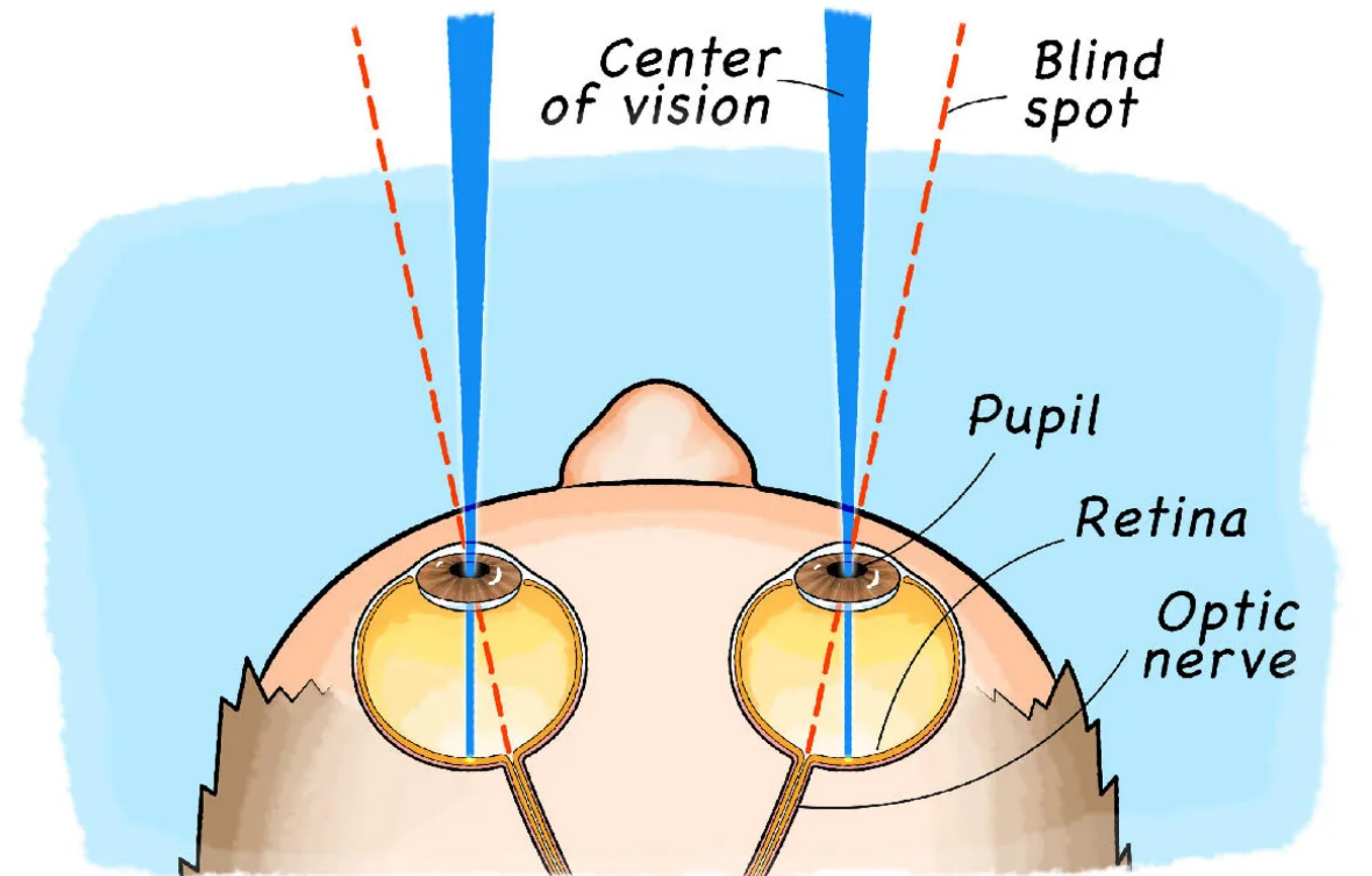
These constants store integer values (e.g., `K_DOWN = 1073741905`), so you can directly pass them to **exp.keyboard.wait** (`keys = [K_DOWN, K_UP, K_LEFT, K_RIGHT]`)

# Exercise 1: Find your blind spot



▲ **FIGURE 9.6**

**The eye in cross section.** Structures at the front of the eye regulate the amount of light allowed in and refract light onto the retina, which wraps around the inside of the eye.



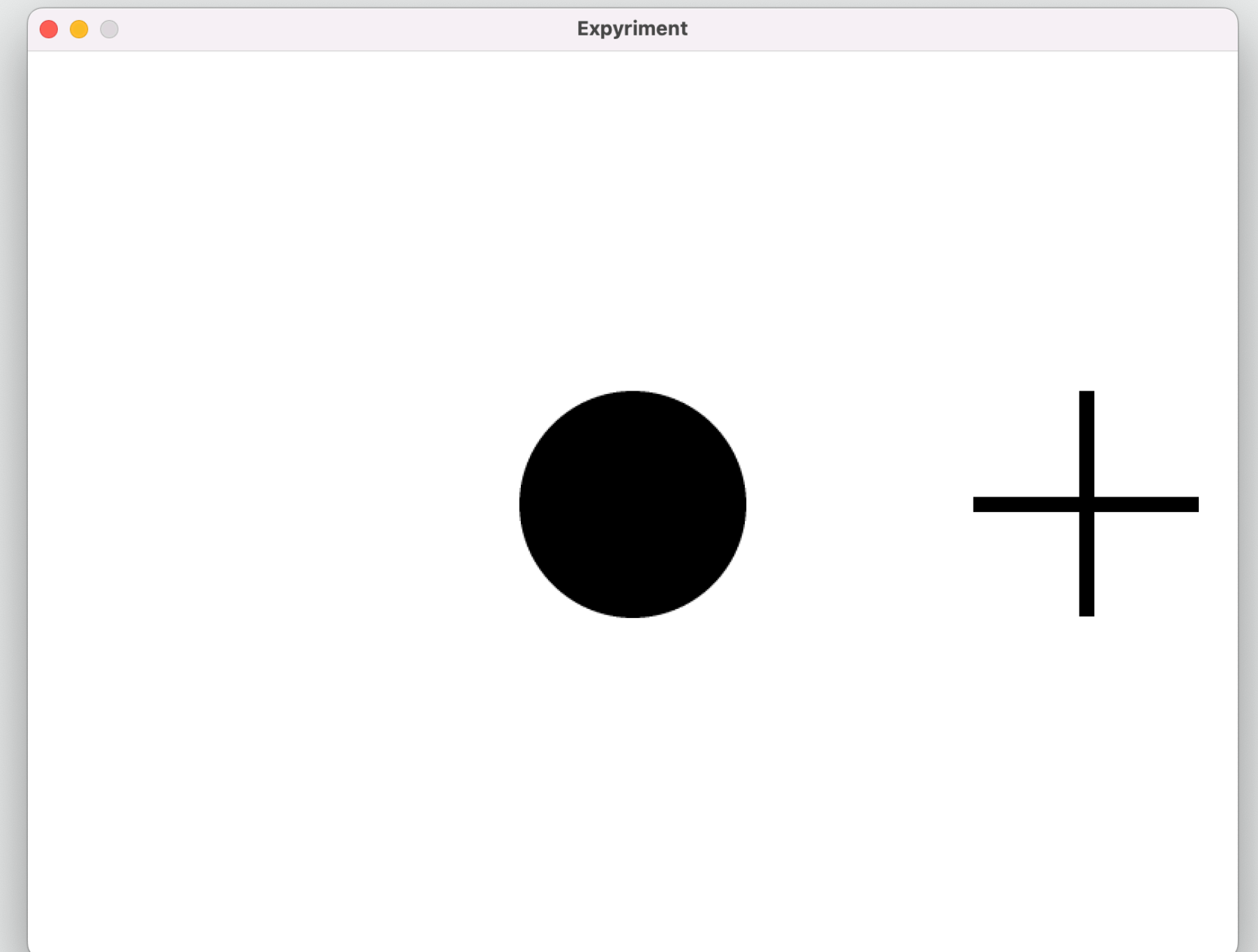
# Exercise 1: Find your blind spot

Change your working directory to  
`Assignments/Week-4/Exercises`

Open `blindspot.py` in VS Code and  
inspect the code

Run it from the Terminal

This is the beginning of a trial meant to locate  
**the blind spot in the left eye**





# Exercise 1: Find your blind spot

Modify the code such that the **position** and **size** of the circle become adjustable (position: keyboard arrows; size: 1 = smaller, 2 = larger)

Add instructions at the beginning of the trial: `stimuli.TextScreen` (which eye to cover, where to fixate, how to adjust the circle, what to do when they're done—press `SPACE` to move on)

Modify `run_trial` so it takes a `side` as input (L/R) and **runs the procedure for the left or right eye** of the subject

# Collecting data

# The datafile

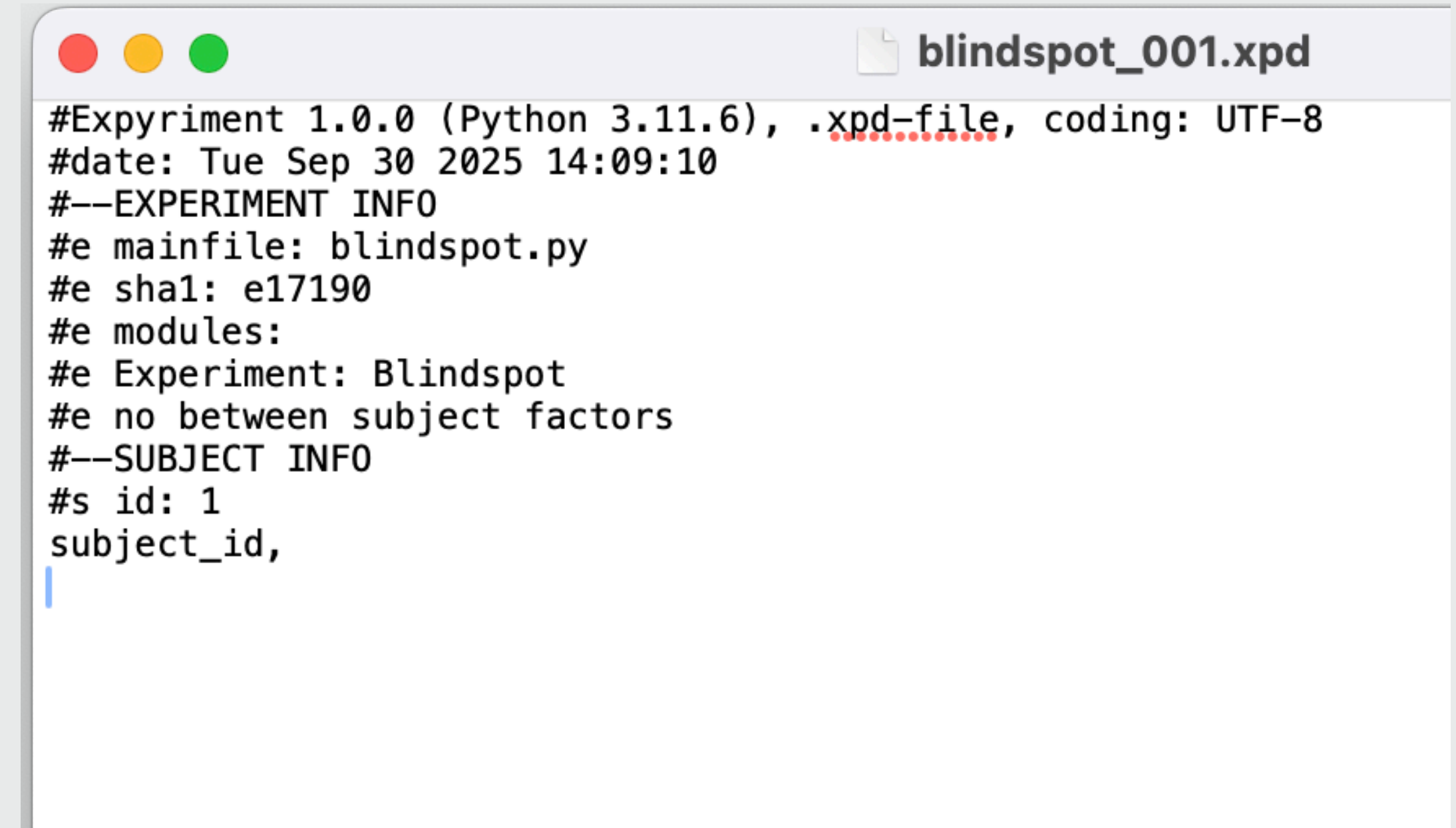
You may have noticed that `expyriment` creates two folders whenever you run a script: `data` and `events`

Navigate to the `blindspot.py` folder and open the `data` subfolder

You should see a file called `blindspot_001.xpd` (if you don't, then you're probably not running your script from the Terminal with the right working dir)

# The datafile

Open `blindspot_001.xpd`  
with a plain text editor (MAC:  
TextEdit; WINDOWS: Notepad)

A screenshot of a text editor window titled "blindspot\_001.xpd". The window has a light gray title bar with three colored window control buttons (red, yellow, green) on the left. The text inside the editor is as follows:

```
#Expyriment 1.0.0 (Python 3.11.6), .xpd-file, coding: UTF-8
#date: Tue Sep 30 2025 14:09:10
#--EXPERIMENT INFO
#e mainfile: blindspot.py
#e sha1: e17190
#e modules:
#e Experiment: Blindspot
#e no between subject factors
#--SUBJECT INFO
#s id: 1
subject_id,
```

The cursor is positioned at the end of the last line, "subject\_id,".

# Writing data to the datafile

In exypriment, the data is a property of the experiment object that can also be customized—conveniently, two lines of code suffice

```
exp = design.Experiment(name="Experiment")

# Pass in a list of string variable names: This will give the names of your columns
exp.add_data_variable_names(["X", "Y", "Z", ...])

# At the end of your run_trial function, store any data you want in exp.data
exp.data.add([x, y, z, ...]) # Can be condition, reaction times, responses, anything!
```

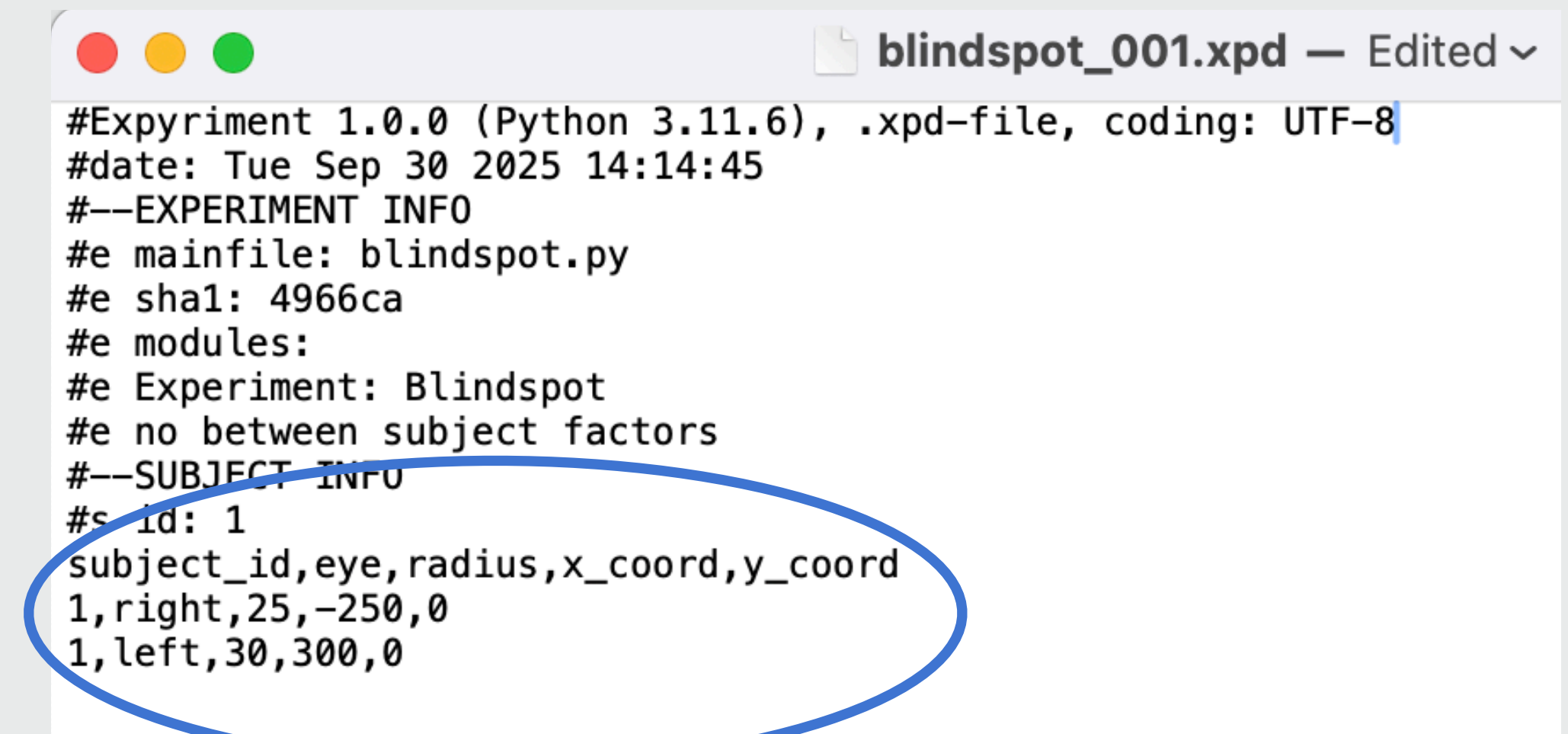
*Note.* No need to add subject ID, expyriment handles that automatically



# Exercise 2A

Modify `blindspot.py` to collect the following data **on each trial**:

- The eye whose blind spot was located
- The radius of the circle
- The coordinates of the circle (separate columns for horizontal and vertical)

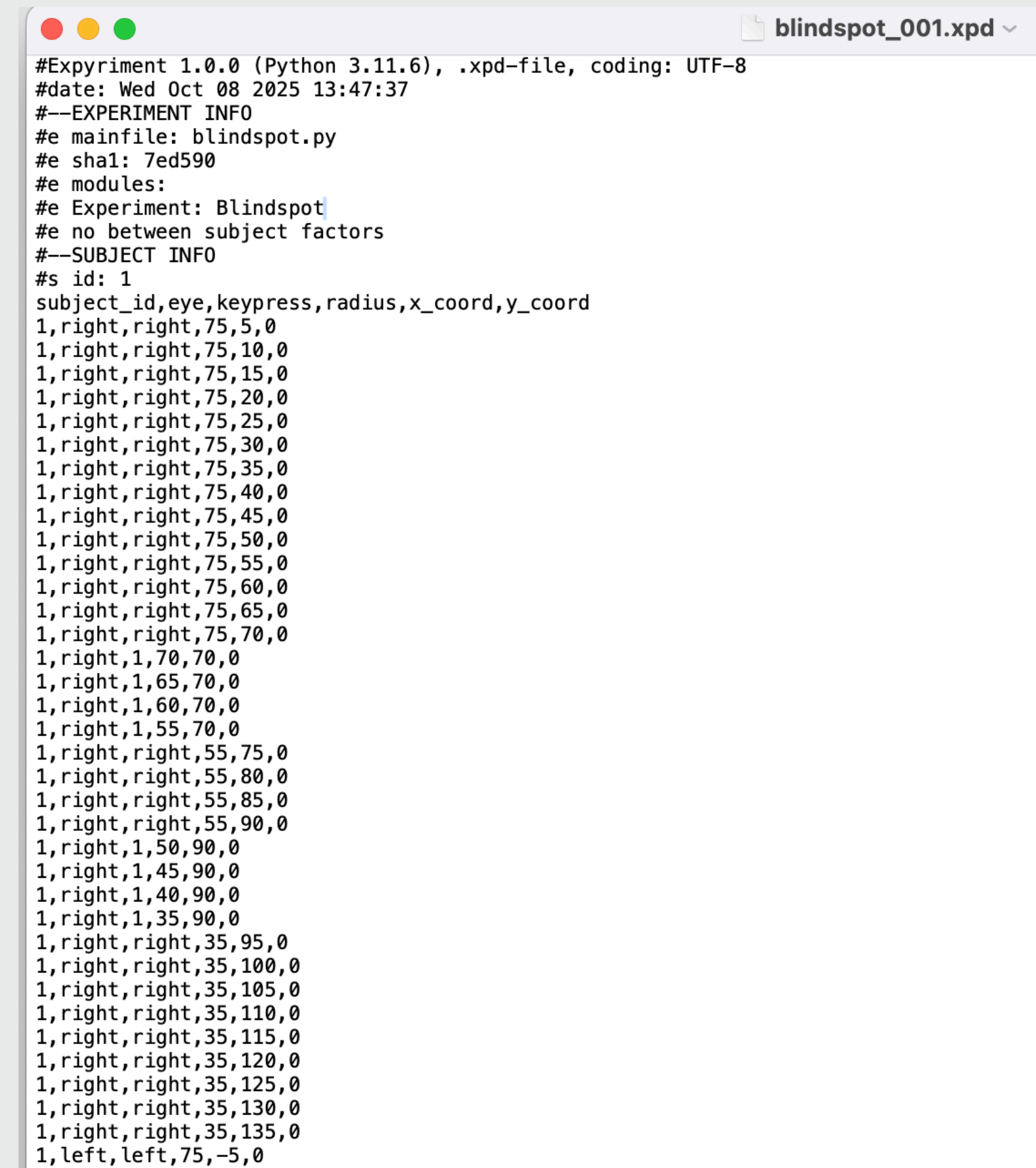


```
#Expyriment 1.0.0 (Python 3.11.6), .xpd-file, coding: UTF-8
#date: Tue Sep 30 2025 14:14:45
#--EXPERIMENT INFO
#e mainfile: blindspot.py
#e sha1: 4966ca
#e modules:
#e Experiment: Blindspot
#e no between subject factors
#--SUBJECT INFO
#s_id: 1
subject_id,eye,radius,x_coord,y_coord
1,right,25,-250,0
1,left,30,300,0
```

# Exercise 2B

Modify `blindspot.py` to collect the following data **on each key press**:

- The trial type (side of the eye)
- The key pressed
- The current radius of the circle
- The current position of the circle



```
#Expyriment 1.0.0 (Python 3.11.6), .xpd-file, coding: UTF-8
#date: Wed Oct 08 2025 13:47:37
#--EXPERIMENT INFO
#e mainfile: blindspot.py
#e sha1: 7ed590
#e modules:
#e Experiment: Blindspot
#e no between subject factors
#--SUBJECT INFO
#s id: 1
subject_id,eye,keypress,radius,x_coord,y_coord
1,right,right,75,5,0
1,right,right,75,10,0
1,right,right,75,15,0
1,right,right,75,20,0
1,right,right,75,25,0
1,right,right,75,30,0
1,right,right,75,35,0
1,right,right,75,40,0
1,right,right,75,45,0
1,right,right,75,50,0
1,right,right,75,55,0
1,right,right,75,60,0
1,right,right,75,65,0
1,right,right,75,70,0
1,right,1,70,70,0
1,right,1,65,70,0
1,right,1,60,70,0
1,right,1,55,70,0
1,right,right,55,75,0
1,right,right,55,80,0
1,right,right,55,85,0
1,right,right,55,90,0
1,right,1,50,90,0
1,right,1,45,90,0
1,right,1,40,90,0
1,right,1,35,90,0
1,right,right,35,95,0
1,right,right,35,100,0
1,right,right,35,105,0
1,right,right,35,110,0
1,right,right,35,115,0
1,right,right,35,120,0
1,right,right,35,125,0
1,right,right,35,130,0
1,right,right,35,135,0
1,left,left,75,-5,0
```

**Push your work to GitHub**