Expyriment main concepts, Visual illusions

Programming Psychology Experiments (CORE-1)

Barbu Revencu & Maxime Cauté

Session 3 | 24 September 2025

The plan for today

1. **Assignments 1–2** discussion
2. **Expyriment `stimuli`**: Present on-screen *what* you want, *how* you want it and *where* you want it
3. **Hands-on programming**: Visual illusions

Assignment 1 Discussion

Submitted solutions: General

Exercise 6: Check if prime

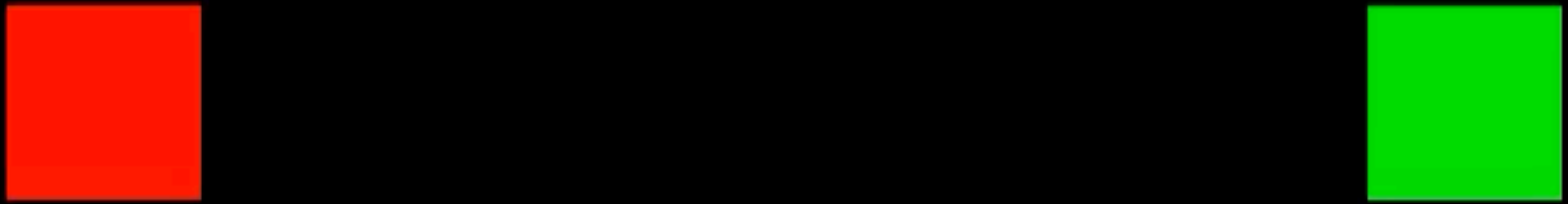
No need to check for divisors all the way up to n: Stopping at the **next integer above** $\sqrt{n}$ suffices (If no divisors up to here, there can’t be any divisors beyond)

Exercise 7: Guess a number in 1–100

Binary search: Computer guesses `(min + max) // 2`, updates `min` or `max` based on user feedback (too high? too low?), then guesses again.

Don’t forget to test edge cases. What happens if we initialize `max` to 100?

Assignment 2 Discussion



Submitted solutions: General

Many solutions obviously copy-pasted from ChatGPT, Claude, etc.

Bad idea—you won’t learn much

Moreover, this included solutions to the *optional* challenge

Why?!?

Many questions on deadlines, penalties, grading

We are not police officers

Submitted solutions: Excerpts

```
expyriment.control.defaults.initialise_delay = 0          # No countdown
expyriment.control.defaults.window_mode = True         # Not full-screen
expyriment.control.defaults.fast_quit = True        # No goodbye message

control.set_develop_mode()   # Does all of the above and more!

# You can also comment this line out when developing
# control.start(subject_id=1)
```

Submitted solutions: Excerpts

```
for frame in range(num_frames):
    draw(shapes)
    exp.clock.wait(x) # x ∈ [5, ..., 17]: Unnecessary, possibly detrimental
```

```
for frame in range(num_frames):
    launcher = stimuli.Rectangle(…) # No need to recreate every frame
    target = stimuli.Rectangle(…)
```

```
def run_launching(temp_gap, space_gap, speed):
     while square2.position[0] < space_gap: # What happens here?
         square2.move((5, 0))

# Michottean launching
run_launching(temp_gap=0, space_gap=-50, speed=5)
```

Submitted solutions: Excerpts

```
offset = 400
speed = 5
frames = 80 # Bad idea to hardcode: frames = offset // speed

for i in range(frames): # If i not used, replace with _
    launcher.move((speed, 0))
```

Robustness: The fewer values you hardcode and the fewer assumptions you make, the better

Python convention: When you need a dummy variable that won’t enter computations, use an underscore (‘_’)

```
for _ in range(frames)
```

Submitted solutions: Excerpts

```
def launching(gap, delay, triggering):
    exp = design.Experiment() # Should not be here
    control.initialize(exp) # Should not be here
    launcher = stimuli.Rectangle(...)
```

Modularize: The `launching` function should only take care of the launching event, not of the experimental sequence

Launching: Problem structure

The constraints that need to be satisfied

1. Launcher moves at some speed…
2. …until it collides with the target…
3. …which then moves the same distance in the same direction

The parameters: Distance, time, speed—one of them is fixed. Which?

Option 1: Compute speed from time

```
to_travel = launcher.distance(target) - launcher.size[0]     # 350 pixels
t = 1                                                        # in seconds
fps = 60                     # frames per second (assuming 60-Hz display)
num_frames = round(t * fps)

speed = to_travel / t                               # 350 pixels / second
step_size = speed / fps                           # 5.8333 pixels / frame

for frame in range(num_frames):
    launcher.move((step_size, 0))

for frame in range(num_frames):
    target.move((step_size, 0))
```

Option 2: Check collision, `time` implicit

```python
step_size = 10

for small_step in range(10000):
    launcher.move((1, 0))
    overlap, _ = launcher.overlapping_with_stimulus(target)

    if overlap:
        launcher.move((-1, 0)) # Gone too far: Backtrack
        if small_step % step_size != 1: draw(shapes) # Avoids double draw
        break

    if small_step % step_size == 0: # Update every 10 small steps
        draw(shapes)
```

Wrapping everything inside a function

```python
def run_trial(length=50, delay=0, gap=0, step_size=10, speed_up=1):
    # Create stimuli
    # Move until collision (add gap if gap ≠ 0)
    # Add delay if necessary: exp.clock.wait(delay)
    # Move target based on speed_up arg: step_size *= speed_up

# One launching, one delay, one gap, one triggering
trials = [{}, {"delay": 500}, {"gap": 50}, {"speed_up": 2}]

for trial_params in trials:
    run_trial(**trial_params)
```

Expyriment main concepts

Expyriment control sequence

```
import expyriment

# PART 1: Global settings go here

exp = expyriment.design.Experiment()
control.initialize(exp)

# PART 2: Stimuli and design (trial & block structure) go here

expyriment.control.start()

# PART 3: Conducting the experiment goes here
# Loop over blocks and trials, present stimuli and get participant input

expyriment.control.end()
```

The *what*: Stimuli generation

Overview

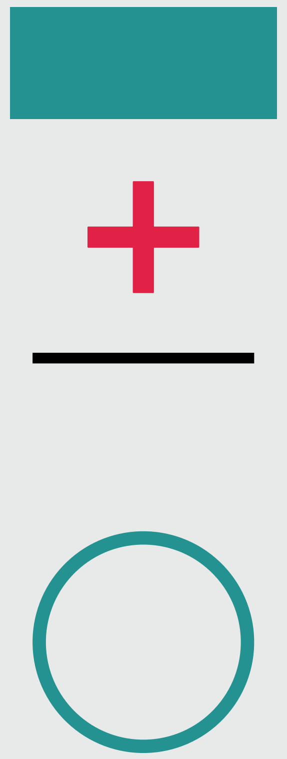
The **`stimuli`** submodule offers a handy way of generating many stimuli common in psychological experiments

You can **customize their properties** by varying the values you pass to the class arguments (size, color, etc.)

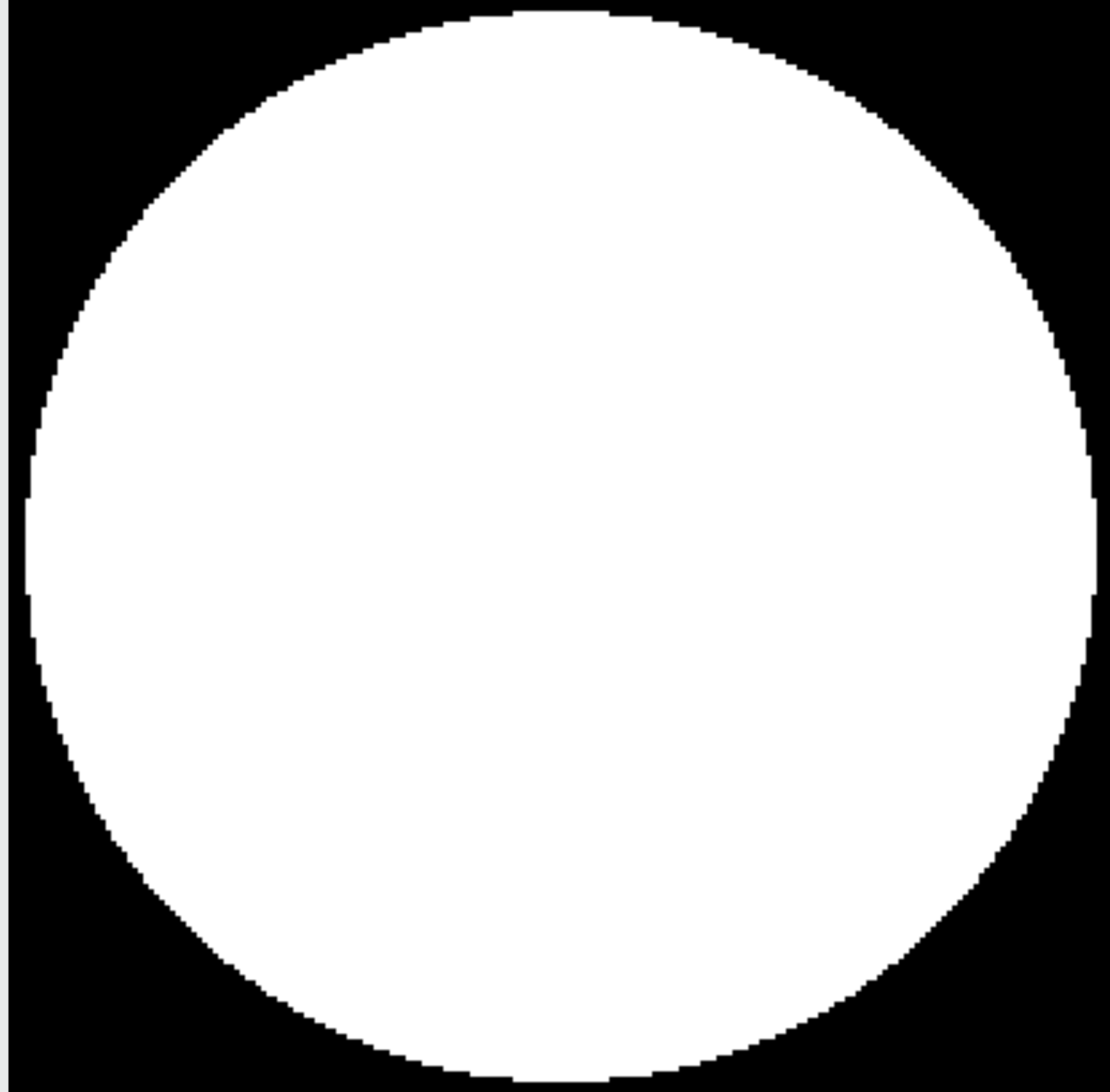
This solves the **what** and the **how** problem in stimulus presentation

Shapes

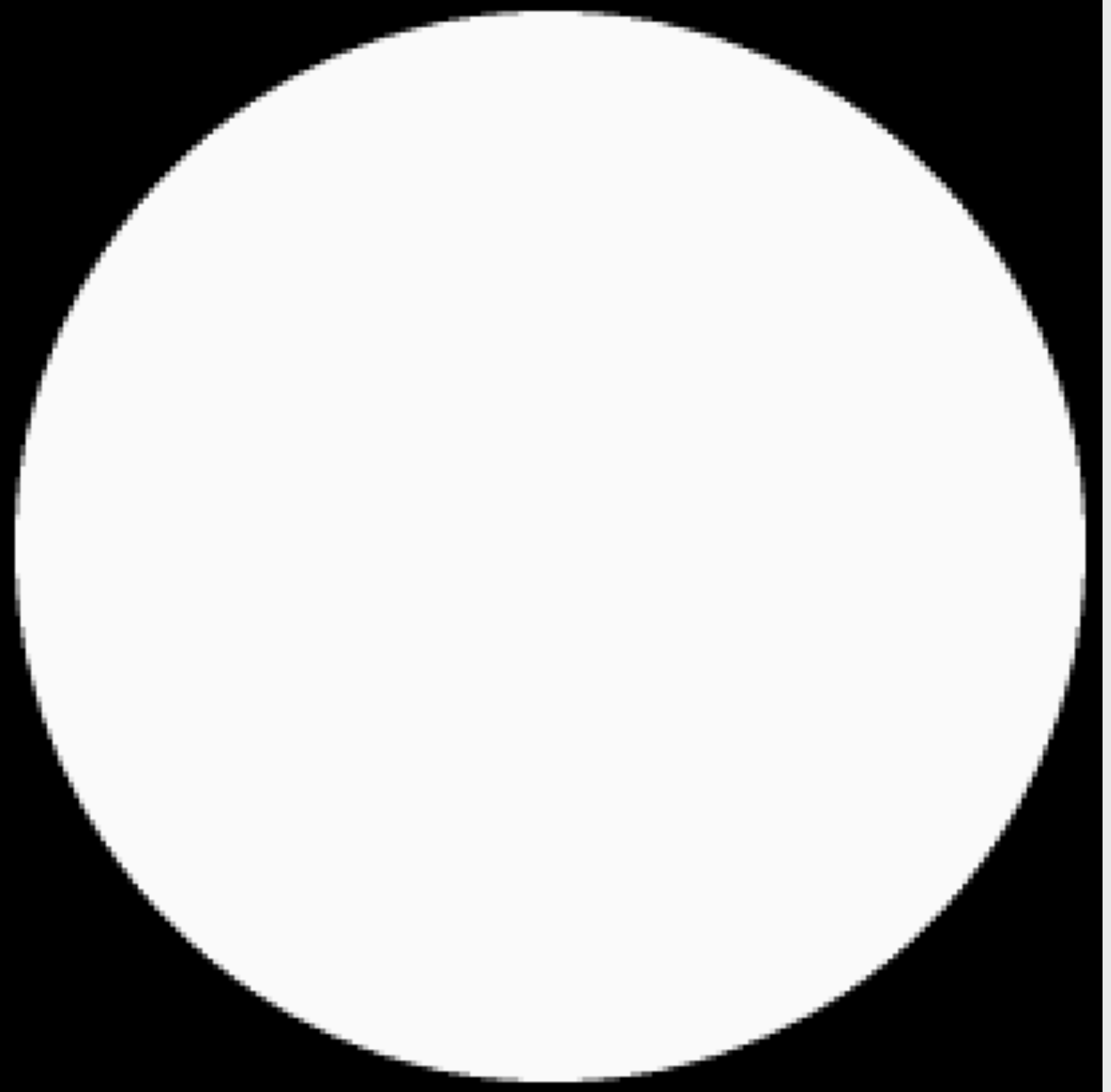
```
# A convenient way of generating common shapes
rectangle = stimuli.Circle(size=(width, height), colour=(R, G, B))
fixation = stimuli.FixCross(size, colour)
line = stimuli.Line(start_point, end_point, colour)

# To create an empty shape, use the line_width parameter
circle = stimuli.Circle(radius, colour, line_width=5)

# If the shape you want does not already have its own class
shape = stimuli.Shape(vertex_list=(...))

# Common colors can be imported from expyriment.misc
misc.constants.C_WHITE, misc.constants.C_GREEN …

# Tip: For smoother edges, set the anti_aliasing parameter to 10
circle = stimuli.Circle(anti_aliasing=10)
```



anti_aliasing = None



anti_aliasing = 10



Text

```
from expyriment import stimuli

# For one-line text stimuli
text = stimuli.TextLine(text, text_size, text_colour)

# For multiline text stimuli
text_multi = stimuli.TextBox(text, size, text_size, text_colour,
background_colour)

# For full-screen text stimuli
text_screen = stimuli.TextBox(heading, text, heading_size, text_size,
text_colour, heading_colour)
```

Images, videos, sounds

```
from expyriment import stimuli

image = stimuli.Picture(filename)  # The path in filename must correspond to an image file on your computer (.png, .jpg, .jpeg, .bmp)

video = stimuli.Video(filename)

audio = stimuli.Audio(filename)
```

The *where*: Stimuli position

On-screen absolute position

To set object positions, pass the desired coordinates in **pixel units**

The coordinates correspond to the shape **center**

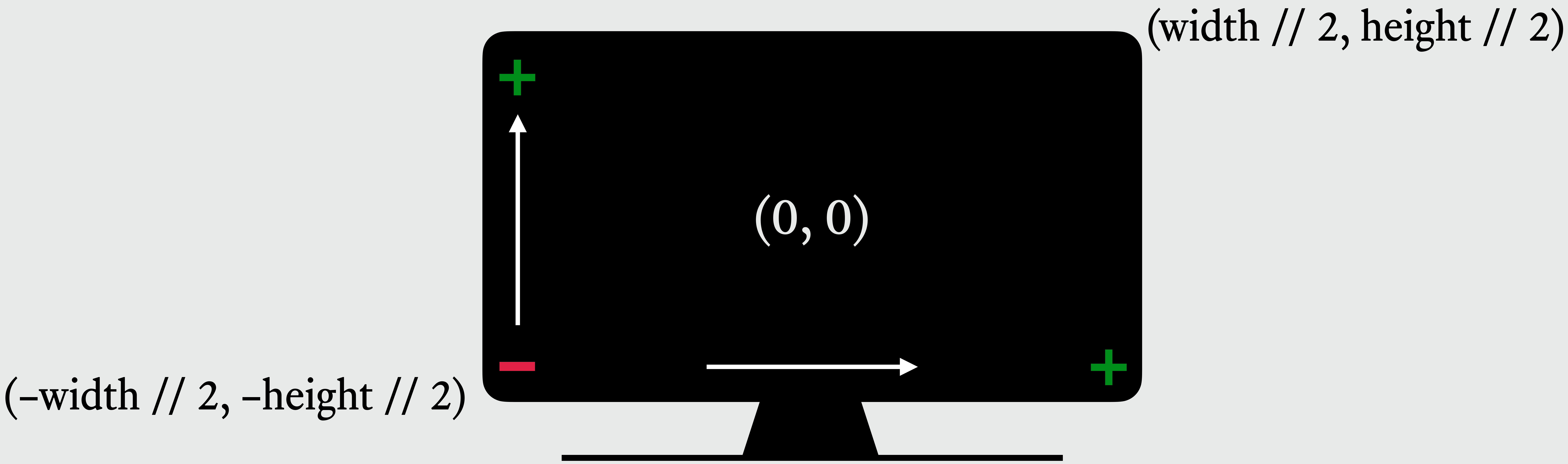
```python
### Three ways of setting the position of a stimulus
# 1. When initializing them
rectangle = stimuli.Rectangle(position=(100, 50))

# 2. After initializing them
rectangle.reposition((-100, 50))

# 3. By moving them (relative to their previous position)
rectangle.move((100, -50))
```

On-screen relative position

Setting absolute positions has limitations

- If you want to present a stimulus ¼-distance away from the left edge, you must know the resolution of your screen and compute it by hand
- The display won't scale with screen size if you run your script from another computer

Obtaining screen coordinates



(width // 2, height // 2)

(0, 0)

(–width // 2, –height // 2)

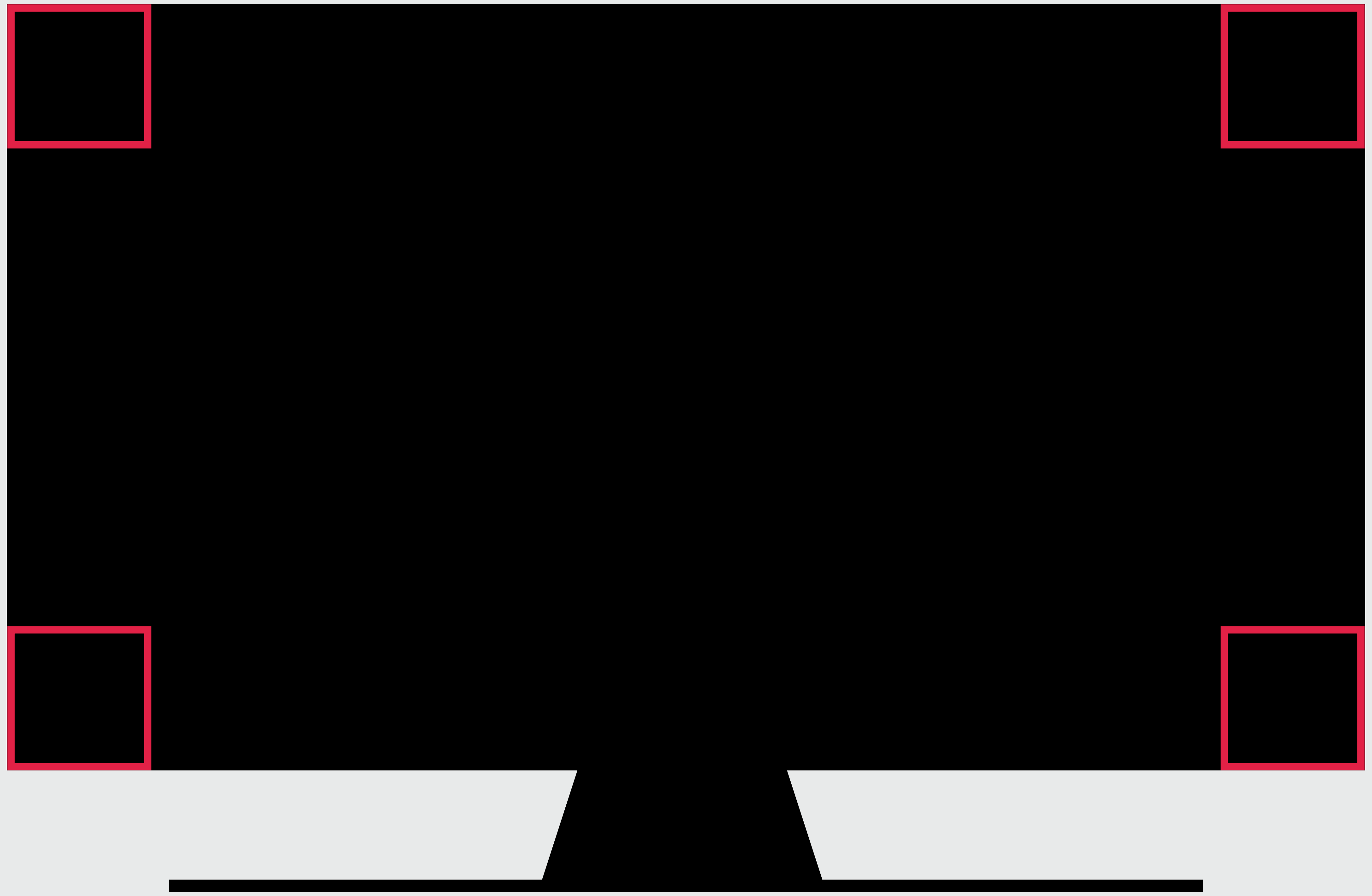
```
width, height = exp.screen.size
```

Exercise 1: `display-edges.py`

Find the screen edges

Present a display of **four fully visible squares with red contours** (square length: ~5% of the screen width, line width: 1 pixel) at the screen edges until a key is pressed

The display must be **independent of screen resolution** (to check this, run w/ and w/o `control.set_develop_mode()`)



Exercise 2: `kanizsa-square.py`

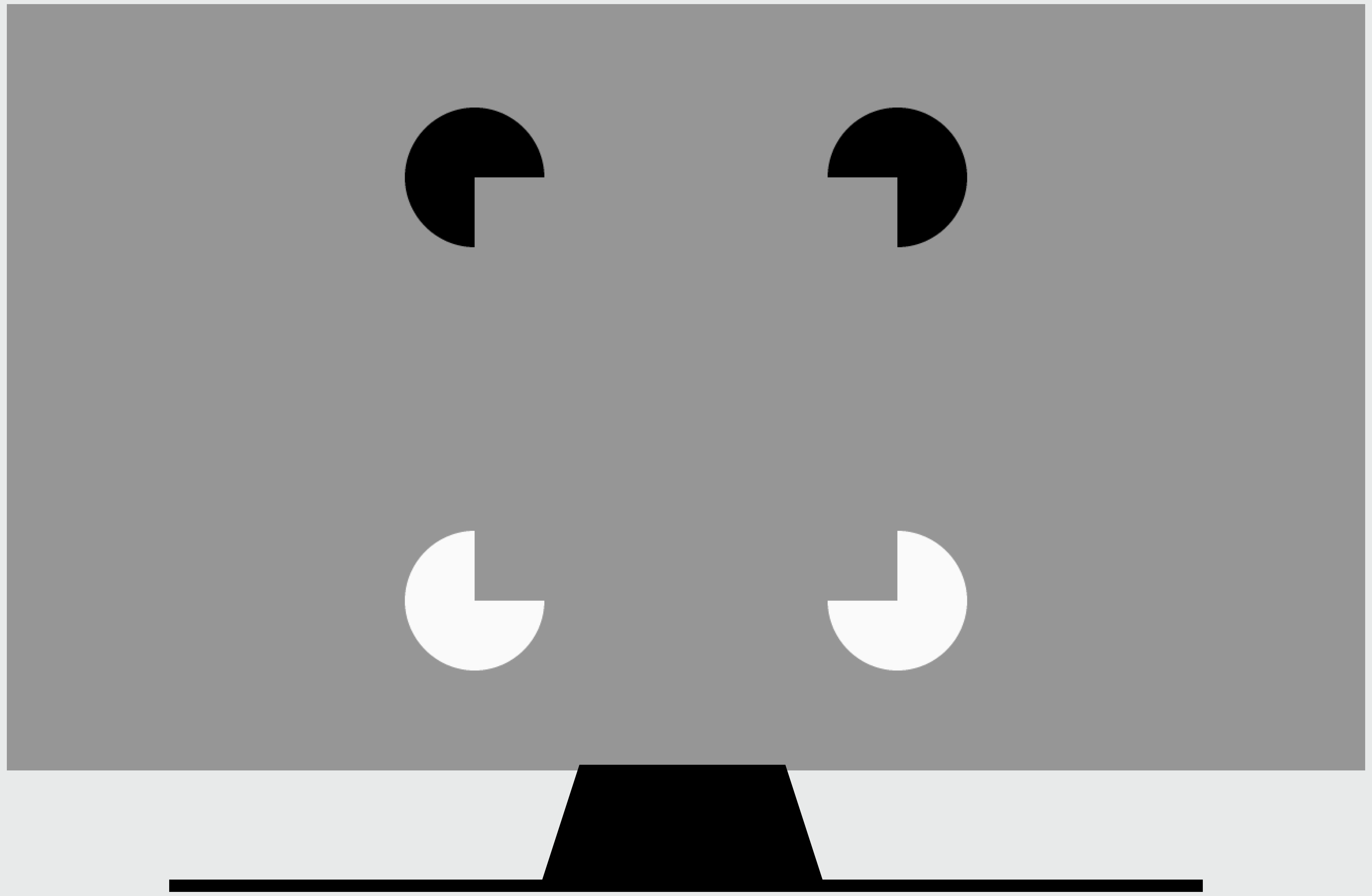
Recreate the Kanizsa square

Display properties:

Square side length = 25% of screen width

Circle radius = 5% of screen width

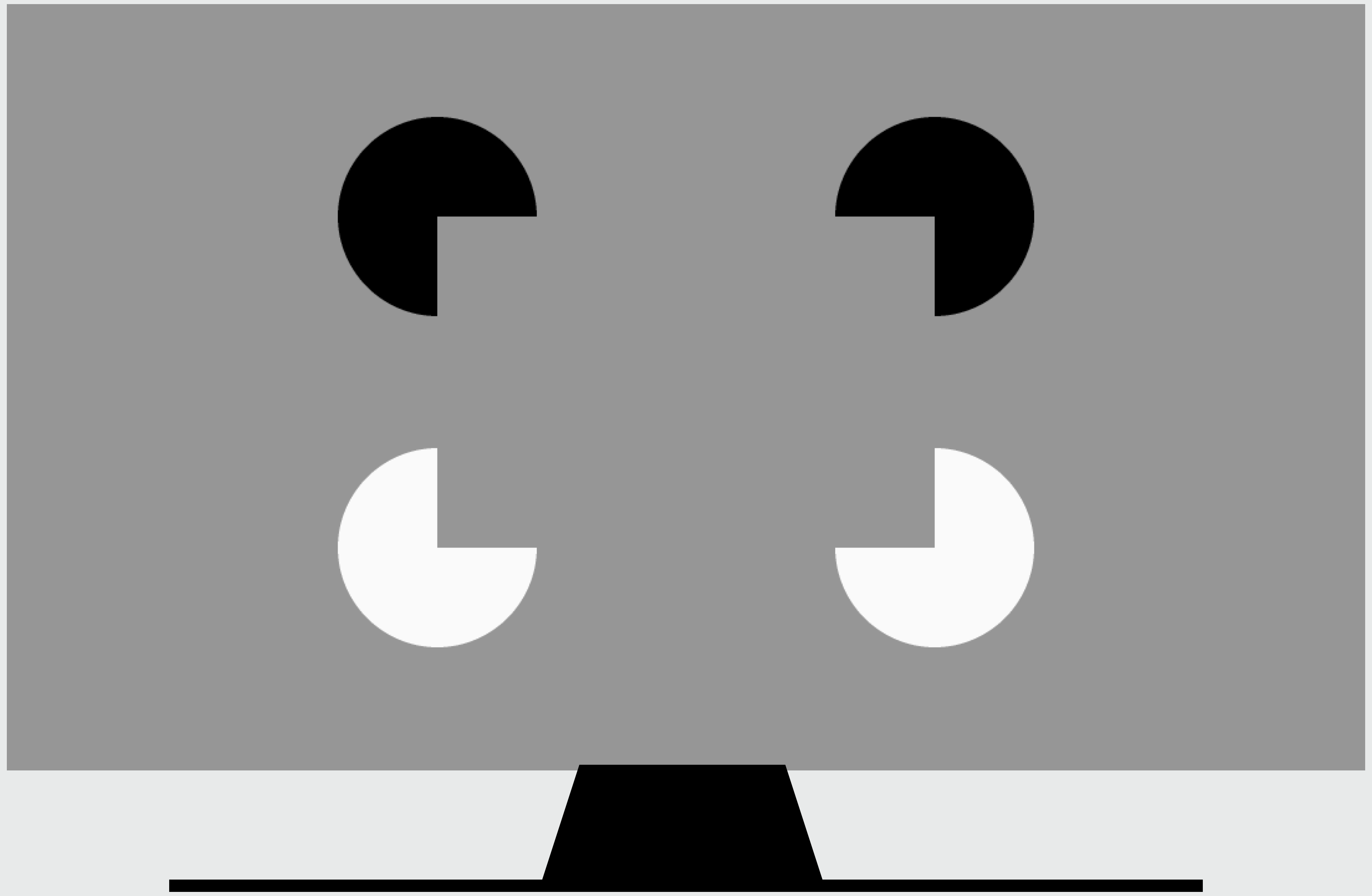
Hint: When initializing the `exp` object, set `background_colour` to `C_GREY`



Exercise 3: `kanizsa-rectangle.py`

Modify your Kanizsa-square code to display a **rectangle** of a given aspect ratio and size

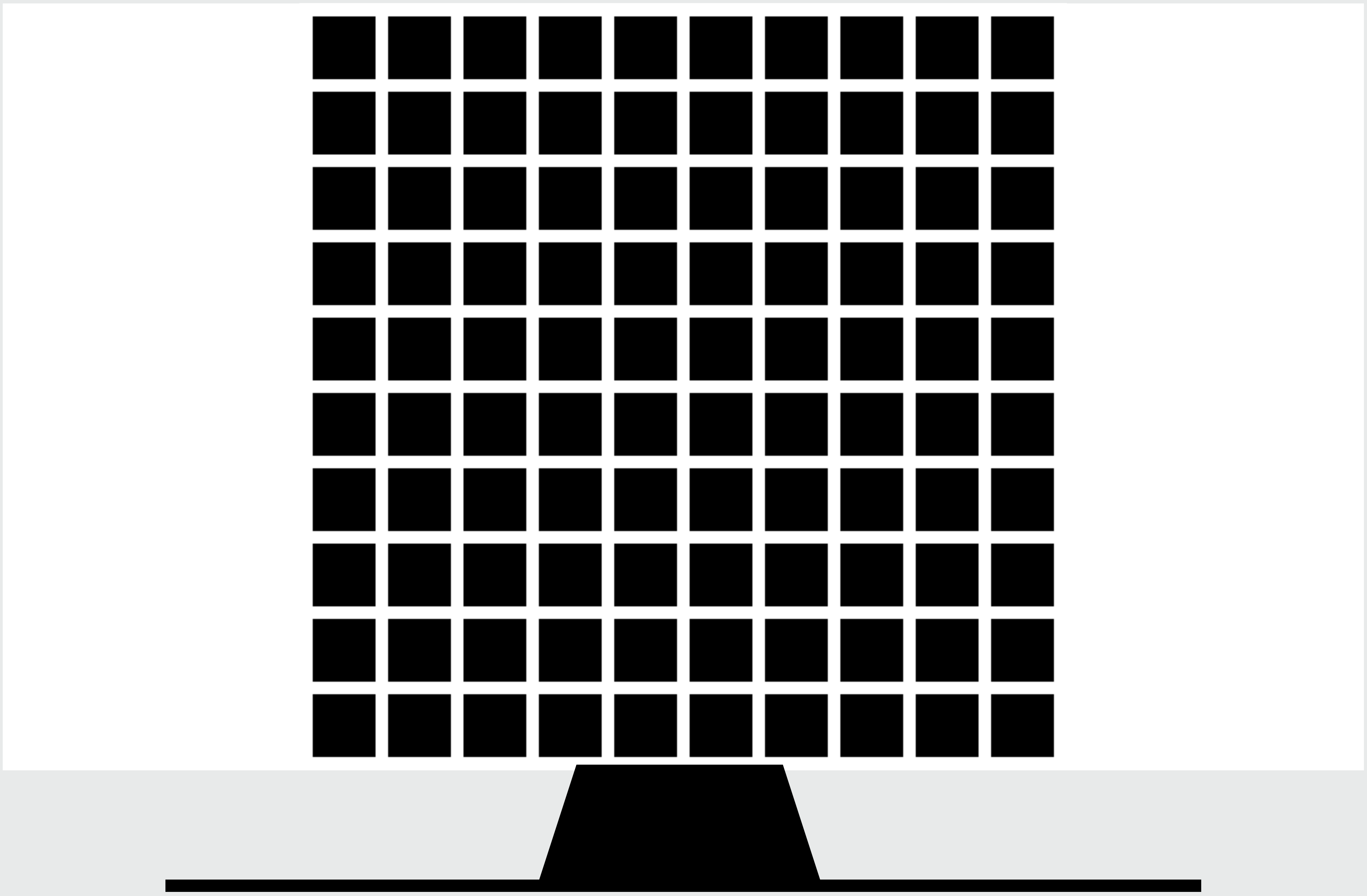
Wrap it inside a **function** whose arguments are the **aspect ratio** of the rectangle and **two scaling factors**: one for the rectangle, one for the circles



Exercise 4: `hermann-grid.py`

Recreate the Hermann grid illusion

The program should have customizable parameters for **square size, space between squares, number of rows, number of columns, square color**, and **background color**



Push your work to GitHub

To-do list

If you haven't already, **join the class Discord** at https://discord.gg/vxwNn6arTm

> If you haven't already, change your display name to your full name: *Raspoutine*, *Sombre*, *Mushroom*, *Featherless Biped*

Check the GitHub-Notes column in this Google Sheet and **update your repository accordingly**: Sam, Nicolas, Leane, Leonor, Amruth, Dario

Homework: Leftover exercises