Programming for psychologists

Lecture 6: PsychoPy

Matthias Nau

What is **PsychoPy** and why do we care?

What is **PsychoPy**?

PsychoPy

Open-source package for running experiments in Python

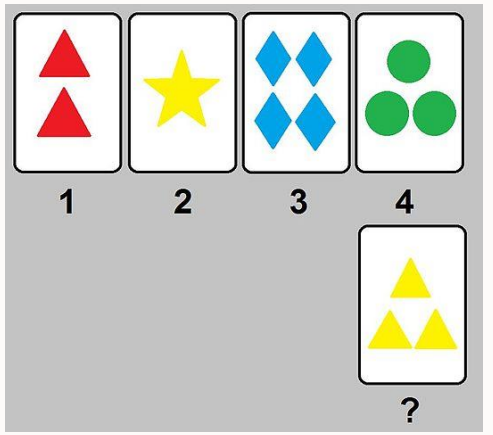
Pavlovia

Provides tools for creating a wide range of experimental paradigms

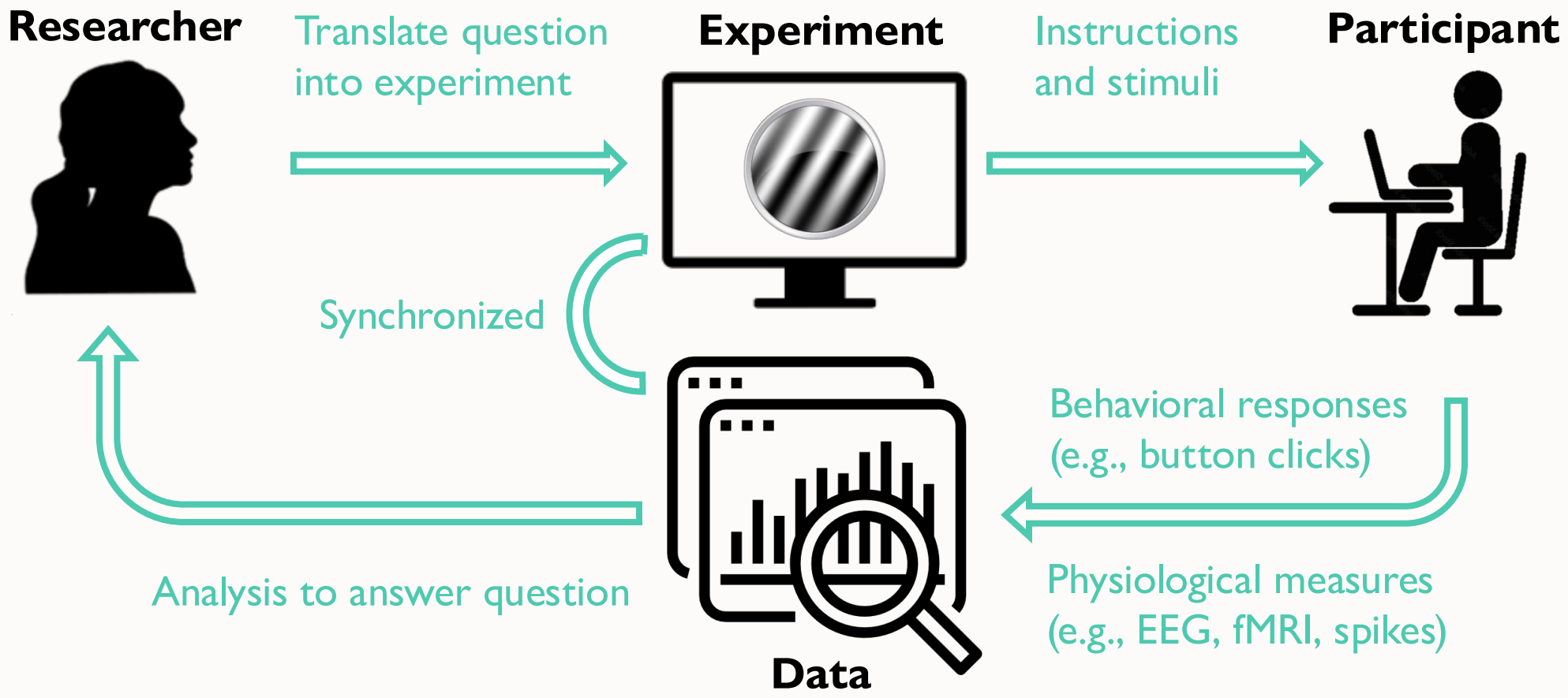
- Presentation of visual, auditory, or tactile stimuli.
- Measuring key presses, joystick movements, eye tracking data etc.
- Simple synchronization with EEG, MEG, or fMRI scanners etc.
- Running online experiments via Pavlovia.org

Very similar to (and inspired by) MatLab's PsychToolBox.

Example experiment



Why do we care about **PsychoPy**?



PsychoPy allows presenting instructions and stimuli to your participants, as well as logging information and synchronizing it with your data for analysis.

There are multiple ways to use **PsychoPy!**

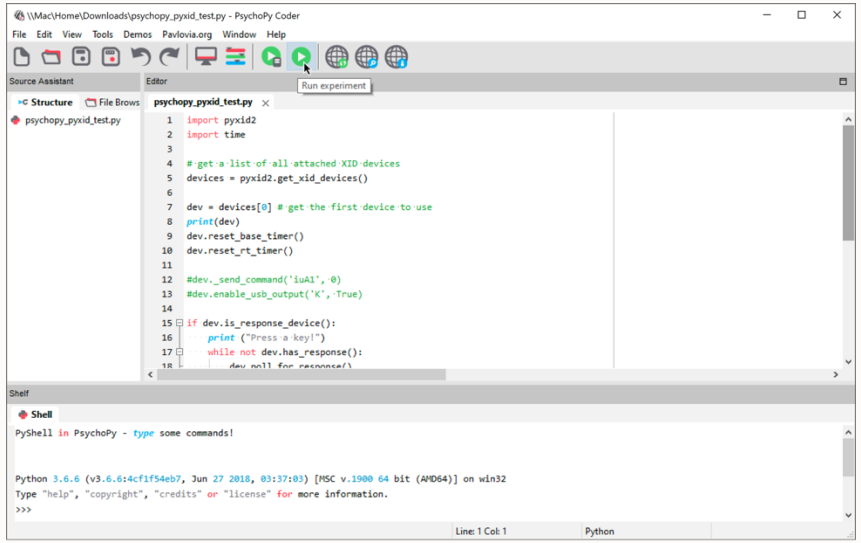
1) PsychoPy Desktop app

2) Conda environment within VScode

We will use this!

A) Builder interface (GUI-based)

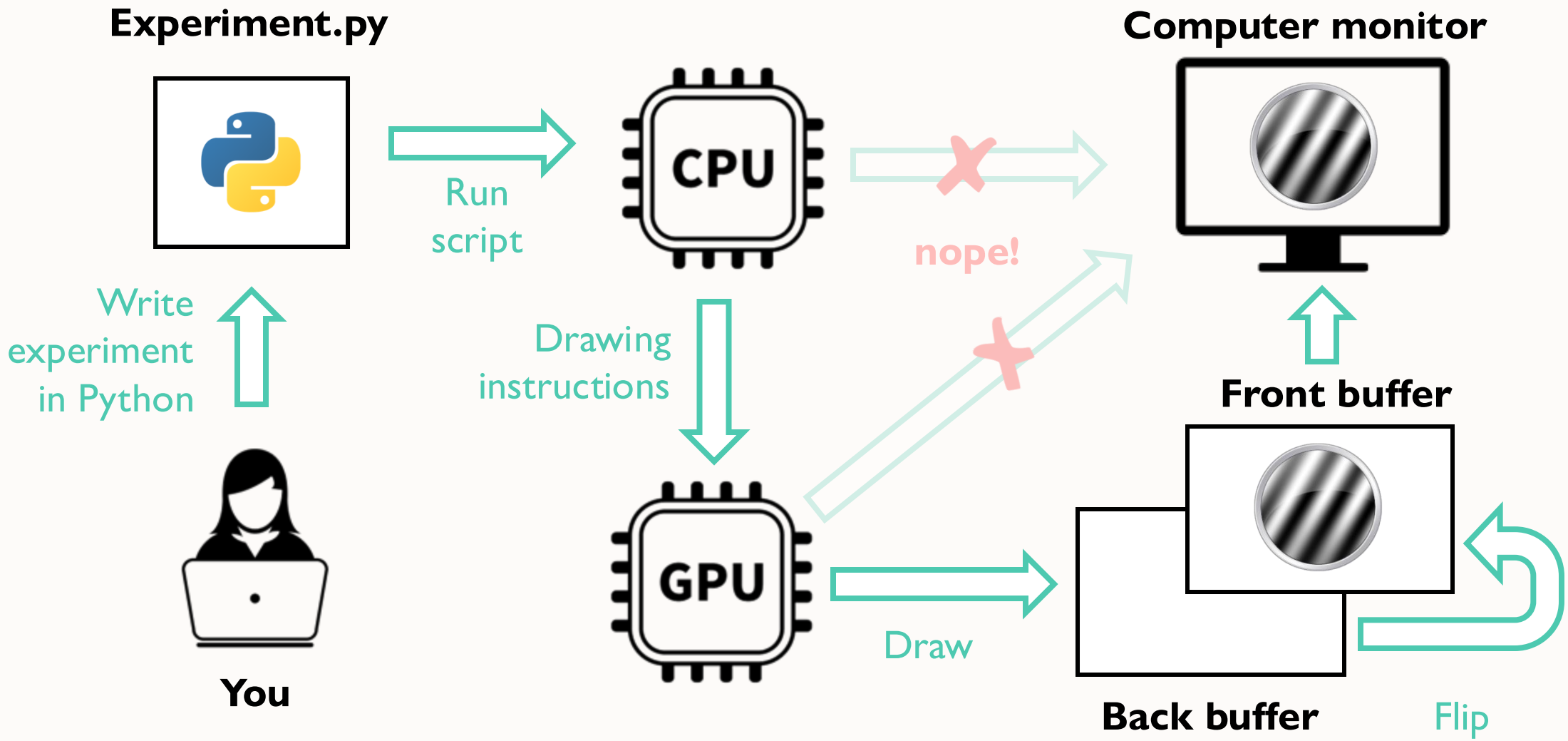
B) Coder interface (Python code)



Note: There are known issues with Apple M-series chips. If your laptop has such a chip, please download the desktop app before the practical. https://www.psychopy.org/download.html

Stimulus presentation essentials

Stimulus presentation essentials



Stimulus presentation essentials

Why do we need a double-buffer system for stimulus presentation?

- **Simultaneous drawing**
  Drawing directly to the screen would mean showing different things at different times.
  The buffer system ensures that everything is shown at once.

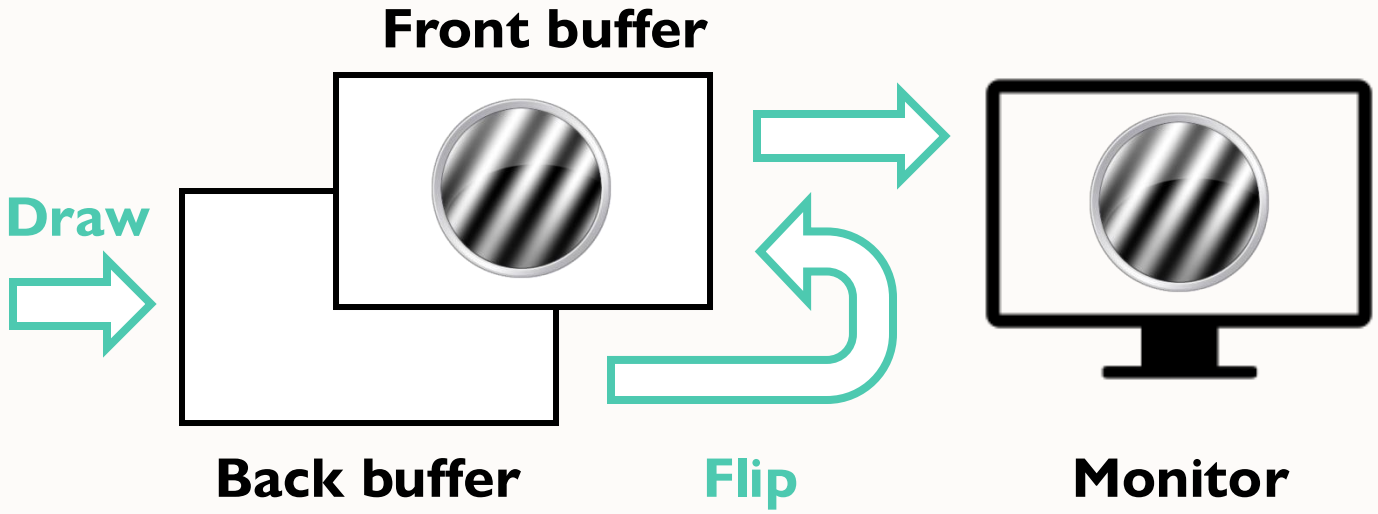
- **Precise timing and syncronization**
  Each frame has an exact time stamp that can be linked to the measured data.

- **Consistency across frames**
  A stimulus looks the same each time it is shown.

- **Off-screen rendering**
  The back buffer allows rendering all stimuli off-screen before displaying them,
  ensuring smooth, flicker-free presentation.

Stimulus presentation essentials

When is the buffer content shown on the screen?



Stimulus is shown at the next refresh time (depends on the refresh rate of your monitor)

This moment is often called the **flip time stamp**.

Refresh rate (also called Frame rate)

- Temporal resolution of your stimulus
- Limited by monitor, typically 60-120Hz
- Beware of frame drops!

What is a frame drop?

Drawing and flipping are not finished at the refresh time, old stimulus remains visible for another frame, and stimulus lags.

Stimulus presentation in PsychoPy

Stimulus presentation in PsychoPy

Different sensory modalities have different PsychoPy modules.
We will focus on the **visual module**, but they all work very similarly.

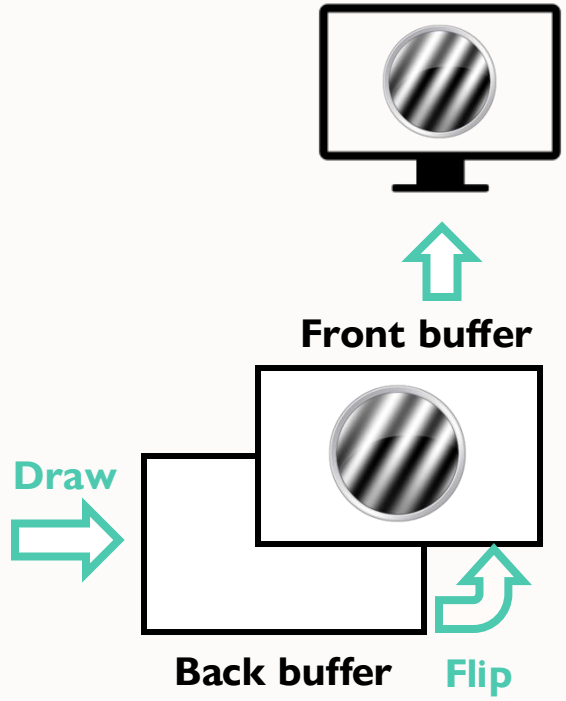
For visual stimulus presentation, the following function is absolutely essential:

- **Visual.Window** Creates an **empty canvas** (typically called "win") on which all stimuli will be displayed, and it controlls which monitor is used, its units etc.

The following functions then allow you to create a wide range of visual stimuli:

- **Visual.TextStim** Create **text** (e.g., for task instructions)
- **Visual.Rect** Create **rectangle** (e.g., for simple stimuli)
- **Visual.Circle** Create **circle** (e.g., for simple stimuli)
- **Visual.Line** Create **line** (e.g., for simple stimuli)
- **Visual.GratingStim** Create **grating** with predefined spatial frequency
- **Visual.DotStim** Create **random dot** stimulus (e.g., for visual-motion experiments)
- **Visual.ImageStim** Prepare **picture** to be shown
- **Visual.MovieStim** Prepare **video clip** to be shown

Stimulus presentation in PsychoPy



The functions of the Visual module allow you to create stimuli, but they do not yet display them.

You still need to **draw the stimuli to the back buffer**, and then **flip them to the front buffer**.

You draw stimuli using the following function:

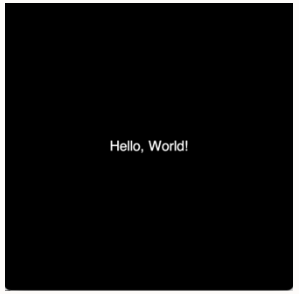
- ***x*.draw()** **Draw** to back buffer. Replace the **x** with stimulus variable name, e.g., my_text.draw(). You can draw multiple stimuli before flipping.

Once drawing has finished, you can flip everything to the front buffer with the following function:

- **win.flip()** **Show** everything you have drawn all at once at the next monitor refresh

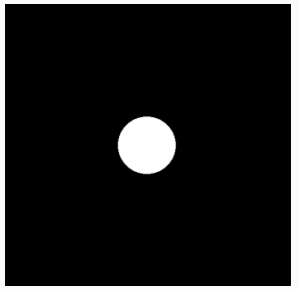
Stimulus presentation in PsychoPy

```python
from psychopy import visual, core

# Set up a window
win = visual.Window(size=(500, 500), color="black")

# Stimulus presentation
text = visual.TextStim(win, text="Hello, World!", color="white") # make stimulus
text.draw() # draw to back buffer
win.flip() # display at next monitor refresh time
core.wait(1) # wait

image = visual.ImageStim(win, image="/Users/matthiasnau/Desktop/VU.png")
image.draw()
win.flip()
core.wait(1)

circle = visual.Circle(win, radius=0.2, fillColor="white")
circle.draw()
win.flip()
core.wait(5)

grating = visual.GratingStim(win, sf=10, ori=45)
grating.draw()
win.flip()
core.wait(5)

# Close the window
win.close()
core.quit()
```

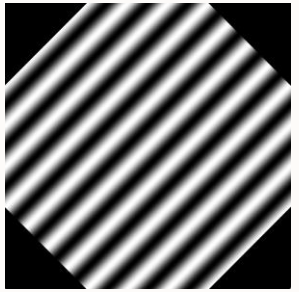


Text for 1s

Picture for 1s



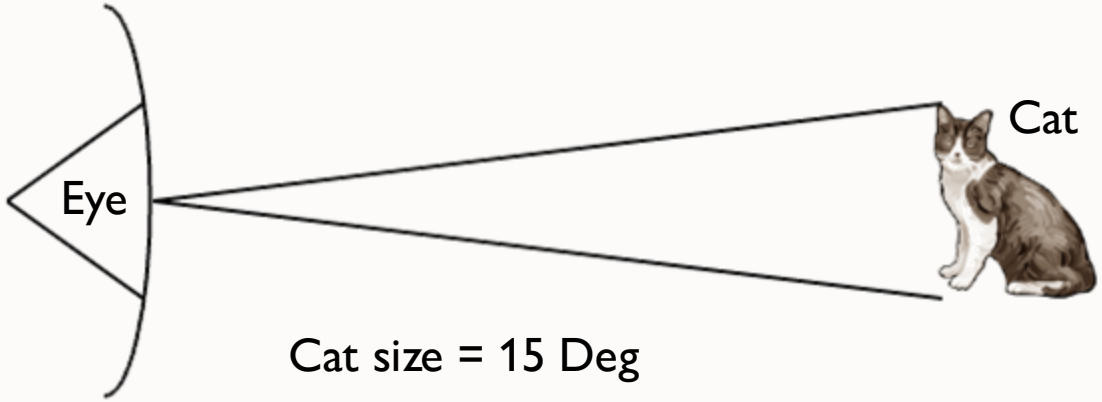
Circle for 5s

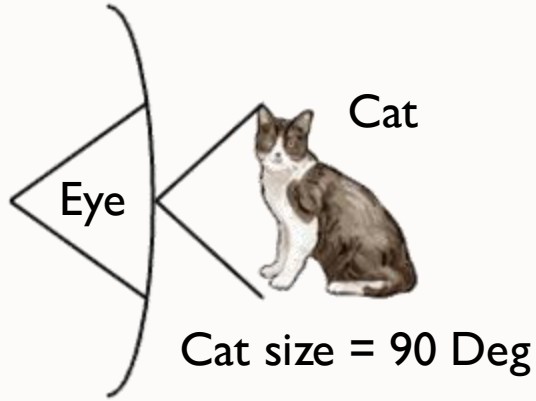


Grating for 5s

Stimulus presentation in PsychoPy

The size of visual stimuli is typically measured in visual degrees, not Pixel or Millimeter.





Visual degrees take the viewing distance into account (i.e., the eye-to-screen distance)

1 Visual Degree equals 1 Centimetre at a viewing distance of 57cm (approximately your arm length)

The following formula allows you to compute the size of your stimuli in degrees:
Visual degrees = 2 · tan-1((size in mm / 2) / distance in mm)

Stimulus presentation in PsychoPy

Two ways of controlling stimulus size in PsychoPy

1) You compute the size of your stimuli in visual degrees using the aforementioned formula. Then, you adjust your code such that the stimuli have the desired size.

2) In the Desktop app, you can let PsychoPy do the job:
"**Tools**" → "**Monitor Center**" → "**New…**" → **Enter Monitor settings** → "**Save**"

   In your code, refer to those settings (e.g., "myMonitor") when setting up the window

```
1  from psychopy import visual, core, monitors
2
3  # Define monitor settings with known viewing distance and screen size
4  mon = monitors.Monitor('myMonitor', width=53.34, distance=60)  # Width & distance in cm
5
6  # Set up a window with units in degrees
7  win = visual.Window(size=(500, 500), units="deg", monitor=mon)
```

Stimulus presentation

Presenting many stimuli in a sequence using a for loop.

Let's build an experiment with 10 trials

Each trial consists of 1 image presented for 2sec, and a gray screen for 1 sec

```python
from psychopy import visual, core, monitors

# Monitor settings and window setup
monitor = monitors.Monitor("myMonitor", width=53.34, distance=60)
win = visual.Window(size=(800, 600), color="gray", units="deg", monitor=monitor)

# List of image file paths (replace these with your actual image file paths)
image_files = [
    "image1.jpg", "image2.jpg", "image3.jpg", "image4.jpg", "image5.jpg",
    "image6.jpg", "image7.jpg", "image8.jpg", "image9.jpg", "image10.jpg"]

# Create an ImageStim object
image_stim = visual.ImageStim(win, size=5)  # 5 visual degrees

# Loop over images
for img_path in image_files:
    # Set the image for the current trial
    image_stim.image = img_path

    # Display the image
    image_stim.draw()
    win.flip()
    core.wait(2)  # 2 seconds

    # Show blank gray screen between images
    win.flip() # flip empty buffer (grey)
    core.wait(1)  # 1 second

# Close window
win.close()
core.quit()
```

Data acquisition and synchronization

Data acquisition with PsychoPy

PsychoPy allows you to **record a wide range of behavioral responses** of your participants, which are an essential component of any dataset you collect.

Key presses

- event.waitKeys() — Waits for a single key press and record
- event.getKeys() — Continuously monitor key presses (e.g., record text)

Mouse clicks

- mouse.getPressed() — Record when a mouse button was clicked
- mouse.getPos() — Record the mouse position on the screen

Joystick movements

- Joystick.getX() — Get X coordinate
- Joystick.getY() — Get Y coordinate

Specialized packages further add **eye tracking**, **audio recordings**, and **physiological data**.

Synchronization

Your data acquisition needs to be temporally aligned with the experiment
(e.g., linking a stimulus to a neural signal requires knowledge of the precise timing of both)

Synchronization is typically achieved by **sending triggers** to the device that records your data (e.g., eye-tracking computer), or by **receiving triggers** from it (e.g., MRI scanner)

Example 1: Send trigger through parallel port

```
from psychopy import parallel

# Initialize the parallel port (on windows at port: 0x0378)
port = parallel.ParallelPort(address=0x0378)

# Send a trigger value (e.g., 1) at stimulus onset
port.setData(1)

# Set back to 0 after a short delay
core.wait(0.01)
port.setData(0)
```

Example 2: Receive trigger as button click

```
# record triggers from MRI scanner
from psychopy import event

# Wait for scanner trigger (e.g., button press "5")
scanner_trigger = event.waitKeys(keyList=["5"])
```

Standard in fMRI:
MRI Scanner starts experiment by sending a trigger as button click

Trial randomization and logging

Trial randomization

Perception, task performance, and neural activity observed in the **current trial** all typically depend on what has been tested in the **previous trial**.

We control for such **order effects** by randomizing (or counterbalancing) the order of tested conditions.

In PsychoPy, the "**TrialHandler**" module allows you to do that easily.

```python
from psychopy import visual, core, data

# Create window
win = visual.Window(size=(800, 600))

# Define conditions
conditions = [
    "/Users/matthiasnau/Desktop/image1.png",
    "/Users/matthiasnau/Desktop/image2.png",
    "/Users/matthiasnau/Desktop/image3.png"]

# Set up TrialHandler
trials = data.TrialHandler(trialList=conditions,
                           nReps=2, method="fullRandom")

# Loop over trials
for trial in trials:
    stimulus = visual.ImageStim(win, image=trial)
    stimulus.draw()
    win.flip()
    core.wait(1)

# Close the window
win.close()
core.quit()
```

Logging

An essential component of your data is the **log file** of your experiment, a recording of what happened when.

A log file includes things like participant information, the date, experimental settings etc.

In addition, the **TrialHandler** makes it easy to **log data of individual trials** (e.g., which condition was tested?)

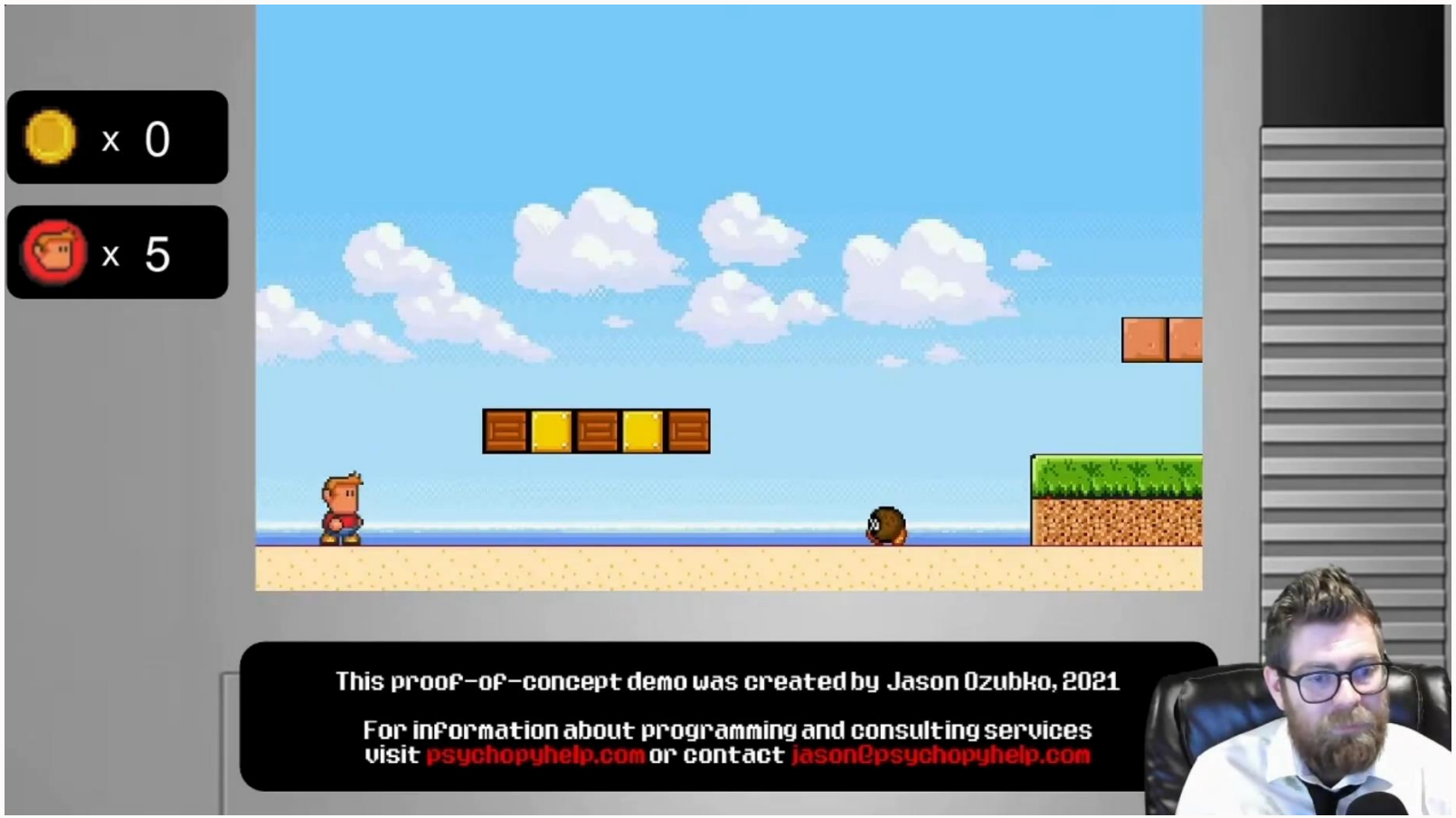
```
12  # Initialize TrialHandler and clock
13  trials = data.TrialHandler(trialList=conditions, nReps=1, method="random")
14
15  # Loop over trials
16  for trial in trials:
17      # show stimulus and record button presses
18      # [...]
19
20      # Add data to the TrialHandler
21      trials.addData("file_path", trial["file_path"])
22      trials.addData("onset_time", onset_time)
23      trials.addData("duration", trial["duration"])
24      trials.addData("response_time", response_time)
25      trials.addData("button_clicked", button)
26
27      # Wait until trial ends
28      core.wait(trial["duration"])
29
30  # Save data as .csv
31  trials.saveAsText("experiment_data_absolute_time.csv", delim=",")
```

Time to install PsychoPy

Note: If you are using PsychoPy inside Conda, create a new virtual environment with Python version 3.8.17!

How far can you take PsychoPy?

How far can you take PsychoPy?



“Supersciencebros”
by Jason Ozubko

Written from scratch in PsychoPy

Check out his tutorials on YouTube!
https://www.youtube.com/c/JasonOzubko

Download game for free: https://psychopyhelp.com/supersciencebros/

Before the next practical, go through these slides again!

Do you know what the following terms mean?

- Back and front buffer
- Drawing
- Flipping
- Monitor refresh rate
- Frame drops
- Stimulus presentation
- Visual module
- Degrees vs. Pixel vs. Millimeter
- Triggers
- Parallel port
- TrialHandler
- Randomization
- Counterbalancing
- Order effects
- Log file

Happy experimenting!