

Experimental Programming and Data Analysis in Python

Lecture 05

PsychoPy Basics

Tao He (何涛)

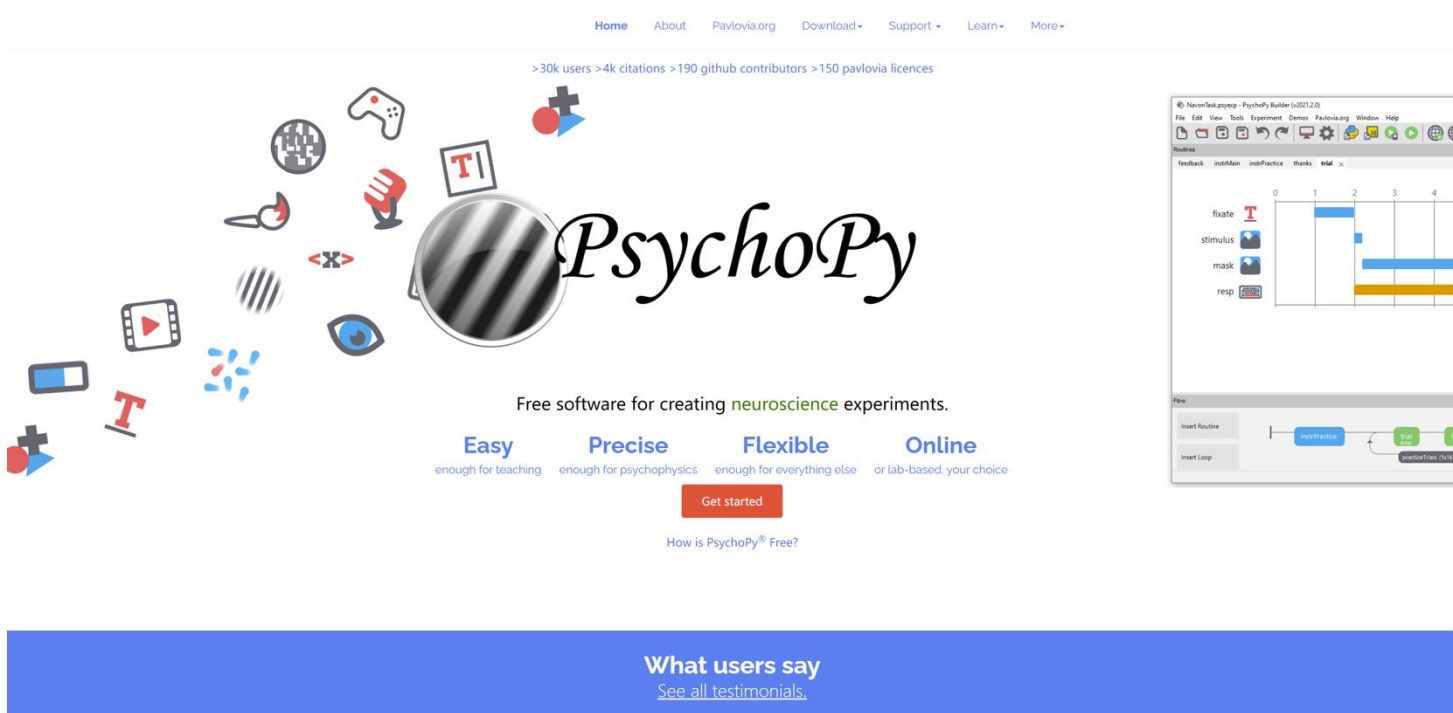
t.he@blcu.edu.cn

School of Psychological and Cognitive Sciences

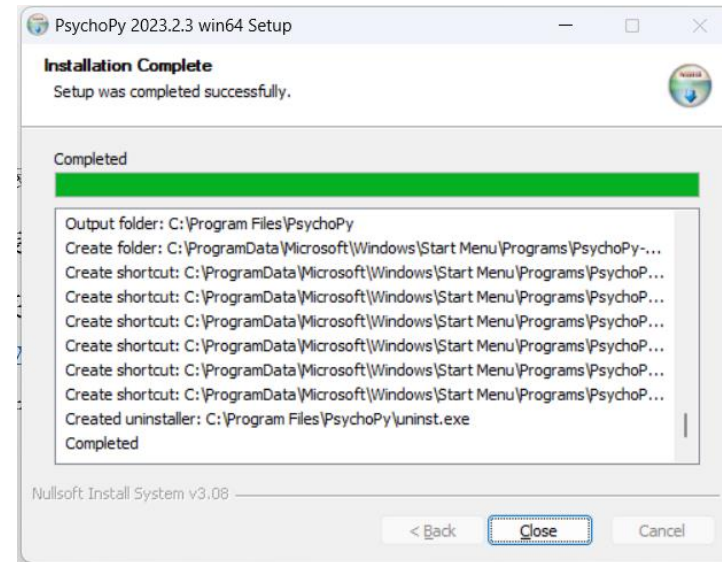
What is PsychoPy?

Psychopy最初是由诺丁汉大学的[Jonathan Peirce](#)教授于2007年基于Python编写的一款心理学实验软件，是免费的开源软件。建议安装当前的独立版本。

<https://www.psychopy.org/download.html>



The screenshot shows the PsychoPy website. At the top, there are navigation links: Home, About, Pavlovia.org, Download, Support, Learn, and More. Below these, it states ">30k users >4k citations >190 github contributors >150 pavlovia licences". The main visual is the PsychoPy logo, which is a stylized 'P' with various icons (game controller, eye, brain, etc.) around it. Below the logo, it says "Free software for creating neuroscience experiments." and lists four features: Easy (enough for teaching), Precise (enough for psychophysics), Flexible (enough for everything else), and Online (or lab-based, your choice). A red "Get started" button is present, along with a link "How is PsychoPy® Free?". At the bottom, there is a blue banner with the text "What users say" and a link "See all testimonials".



PsychoPy Interface

PsychoPy Builder



PsychoPy Coder

The screenshot displays the PsychoPy Coder interface (v2022.2.4). The main window shows a Python script for a 'hello world' program. The script is as follows:

```
1 #!/usr/bin/env python
2 #-*- coding: utf-8 -*-
3
4 """
5 Demo: show a very basic program: hello world
6 """
7
8 # Import key parts of the PsychoPy library:
9 from psychopy import visual, core
10
11 # Create a visual window:
12 win = visual.Window(units="height")
13
14 # Create (but not yet display) some text:
15 msg1 = visual.TextBox2(win,
16     text="Hello world!",
17     font="Open Sans", letterHeight=0.1,
18     pos=(0, 0.2))
19 msg2 = visual.TextBox2(win,
20     text="u00A1Hola mundo!",
21     font="Open Sans", letterHeight=0.1,
22     pos=(0, -0.2))
23
```

Below the script, there is a 'Shell' section showing the output of the program. The output is as follows:

```
##### Experiment ended with exit code 0 [pid:49603] #####
356.4093 WARNING Monitor specification not found. Creating a temporary one...
356.5590 INFO Loaded monitor calibration from ['2019_12_26 23:00']
```

The status bar at the bottom indicates 'Line: 1 Col: 1' and 'Python'.

PsychoPy – where to get help

PsychoPy Manual <https://www.psychopy.org/PsychoPyManual.pdf>

API reference manual <https://psychopy.org/api/index.html>

Reference Manual (API)

Contents:

- `psychopy.core` - basic functions (clocks etc.)
- `psychopy.clock` - Clocks and timers
- `psychopy.session` - for running a session with multiple experiments
- `psychopy.visual` - many visual stimuli
- `psychopy.sound` - for playback and recording of sound
- `psychopy.hardware` - hardware interfaces
- `psychopy.iohub` - ioHub event monitoring framework
- `psychopy.tools` - miscellaneous tools
- `psychopy.app` - the PsychoPy® application suite

What elements do we need in an experiment?

- Collect participant information (name, age, gender, etc.)
- Present stimuli (instruction, visual/auditory stimuli, texts, etc.)
- Collect responses (mouse, keyboard, joystick, etc.)
- Timing accuracy
- Save data

My first PsychoPy script

- Display “Hello world!” on the screen for 3 seconds.
 - demo_hello_world.py
 - demo_hello_world_2.py

Experiment Header

```
>> #!/usr/bin/env python
```

Tell your OS that this programme is using python language

```
>> # -*- coding: utf-8 -*-
```

Character encoding

声明python代码的文本格式是[utf-8编码](#)，
如果不加这个声明，无论代码中还是注释中有中文都会报错。

- Python 3.x 版本中，字符串默认使用 UTF-8 编码
- Python 2.x 版本中，默认的编码方式是 ASCII

```
# Import PsychoPy library that you needed
```

```
>> from psychopy import visual, core
```

Creating a window

```
# Create a visual window:  
>> win = visual.Window(size=(800,600),  
                        units='pix',  
                        fullscr=False,  
                        color=(128,128,128),  
                        colorSpace='rgb255')
```

One of the most important classes from the psychopy package is the **Window** class, which defines the window in which you are going to run your experiment.

Window

A class representing a window for displaying one or more stimuli.

```
class psychopy.visual.Window(size=(800, 600), pos=None, color=(0, 0, 0), colorSpace='rgb',  
backgroundImage=None, backgroundFit='cover', rgb=None, dkl=None, lms=None, fullscr=None, allowGUI=None,  
monitor=None, bitsMode=None, winType=None, units=None, gamma=None, blendMode='avg', screen=0,  
viewScale=None, viewPos=None, viewOri=0.0, waitBlanking=True, allowStencil=False, multiSample=False,  
numSamples=2, stereo=False, name='window1', title='PsychoPy', checkTiming=True, useFBO=False,  
useRetina=True, autoLog=True, gammaErrorPolicy='raise', bpc=(8, 8, 8), depthBits=8, stencilBits=8,  
backendConf=None)
```

[source]

Used to set up a context in which to draw objects, using either `pyglet`, `pygame`, or `glfw`.

The `pyglet` backend allows multiple windows to be created, allows the user to specify which screen to use (if more than one is available, duh!) and allows movies to be rendered.

The `GLFW` backend is a new addition which provides most of the same features as `pyglet`, but provides greater flexibility for complex display configurations.

`Pygame` may still work for you but it's officially deprecated in this project (we won't be fixing `pygame`-specific bugs).

These attributes can only be set at initialization. See further down for a list of attributes which can be changed after initialization of the `Window`, e.g. `color`, `colorSpace`, `gamma` etc.

- Parameters:**
- **size** (array-like of *int*) – Size of the window in pixels [x, y].
 - **pos** (array-like of *int*) – Location of the top-left corner of the window on the screen [x, y].
 - **color** (array-like of *float*) – Color of background as [r, g, b] list or single value. Each gun can take values between -1.0 and 1.0.
 - **fullscr** (*bool* or *None*) – Create a window in 'full-screen' mode. Better timing can be achieved in full-screen mode.
 - **allowGUI** (*bool* or *None*) – If set to *False*, window will be drawn with no frame and no buttons to close etc., use *None* for value from preferences.

<https://psychopy.org/api/visual/window.html#psychopy.visual.Window>

Creating stimuli

```
# Create (but not yet display) some text:  
>> msg1 = visual.TextStim(win, text="Hello world!", pos=(0, 100))  
>> msg2 = visual.TextStim(win, text=u"你好, 世界!", pos=(0, -100))
```

Create a text object
Important method: `.draw()`

TextStim

```
class psychopy.visual.TextStim(win, text='Hello World', font='', pos=(0.0, 0.0), depth=0, rgb=None, color=(1.0, 1.0, 1.0), colorSpace='rgb', opacity=1.0, contrast=1.0, units='', ori=0.0, height=None, antialias=True, bold=False, italic=False, alignHoriz=None, alignVert=None, alignText='center', anchorHoriz='center', anchorVert='center', fontFiles=(), wrapWidth=None, flipHoriz=False, flipVert=False, languageStyle='LTR', draggable=False, name=None, autoLog=None, autoDraw=False)
```

[\[source\]](#)

Class of text stimuli to be displayed in a `Window`

Performance OBS: in general, TextStim is slower than many other visual stimuli, i.e. it takes longer to change some attributes. In general, it's the attributes that affect the shapes of the letters: `text`, `height`, `font`, `bold` etc. These make the next `.draw()` slower because that sets the text again. You can make the `draw()` quick by calling re-setting the text (`myTextStim.text = myTextStim.text`) when you've changed the parameters.

In general, other attributes which merely affect the presentation of unchanged shapes are as fast as usual. This includes `pos`, `opacity` etc.

<https://psychopy.org/api/visual/textstim.html#psychopy.visual.TextStim>

Presenting stimuli

```
# Draw the text to the hidden visual buffer:
```

```
>> msg1.draw()
```

```
>> msg2.draw()
```

```
# Show the hidden buffer--everything that has been drawn since the last win.flip():
```

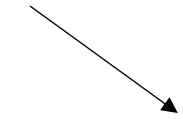
```
>> win.flip()
```

Draw stimuli on the back buffer

Flip the back buffer to the primary surface

Presenting stimuli

Text object



Hello world!

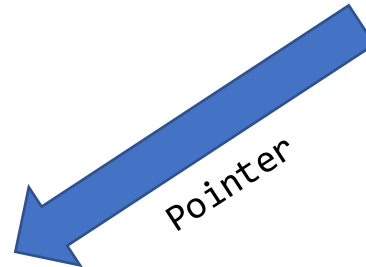
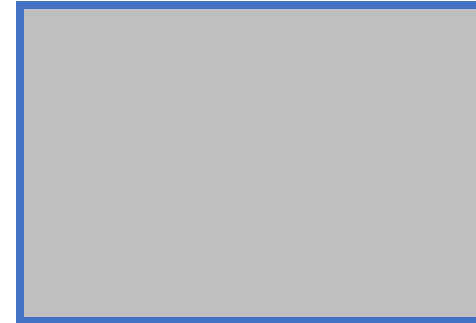
Back buffer



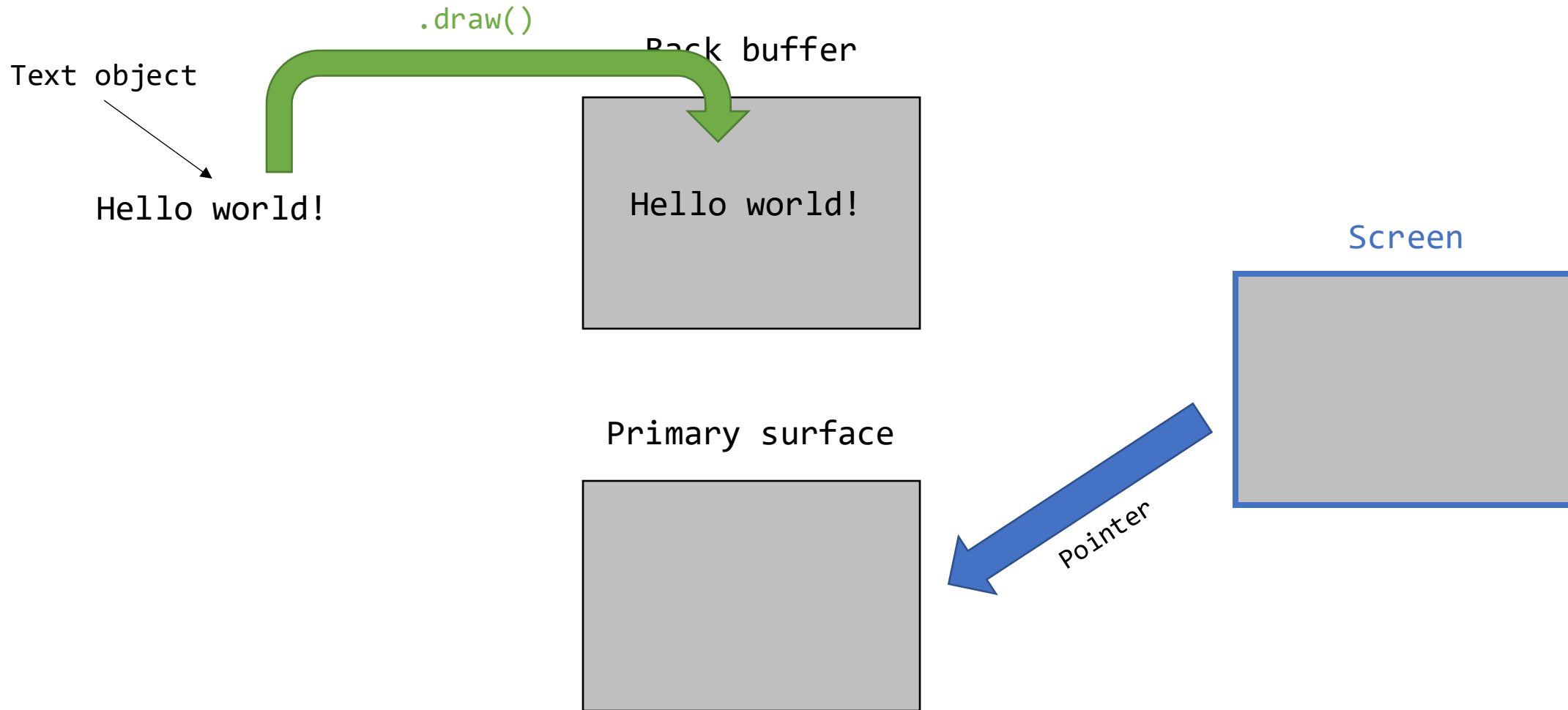
Primary surface



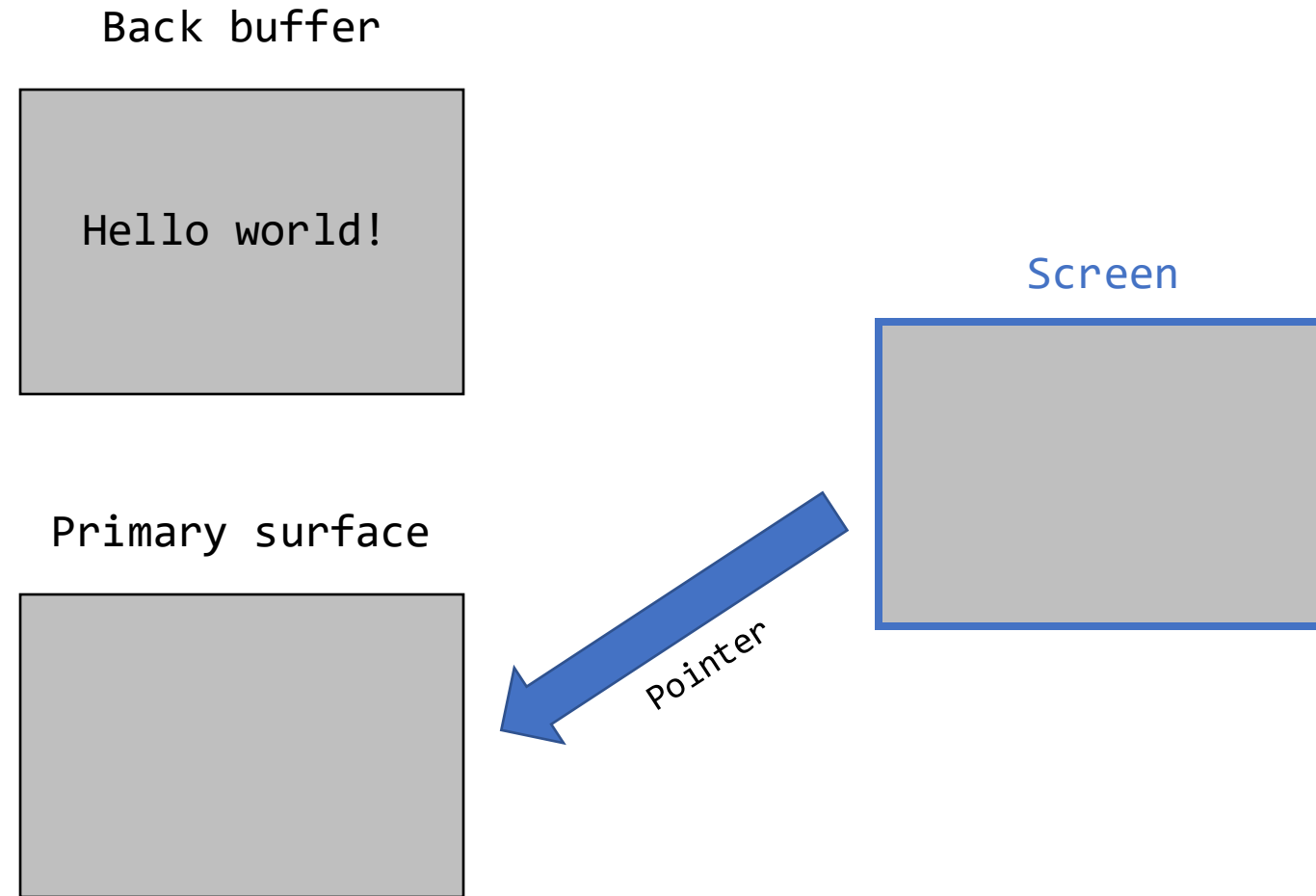
Screen



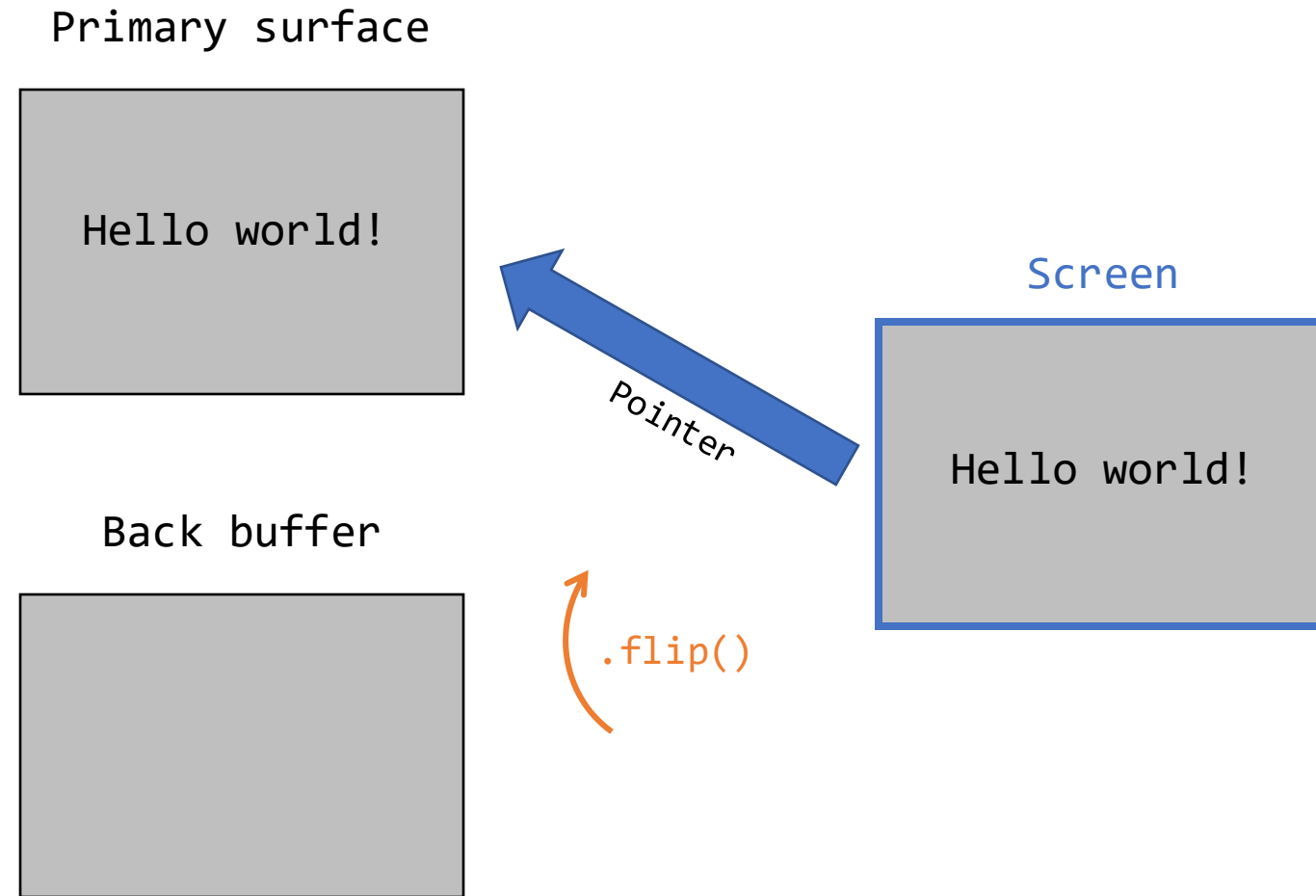
Presenting stimuli



Presenting stimuli

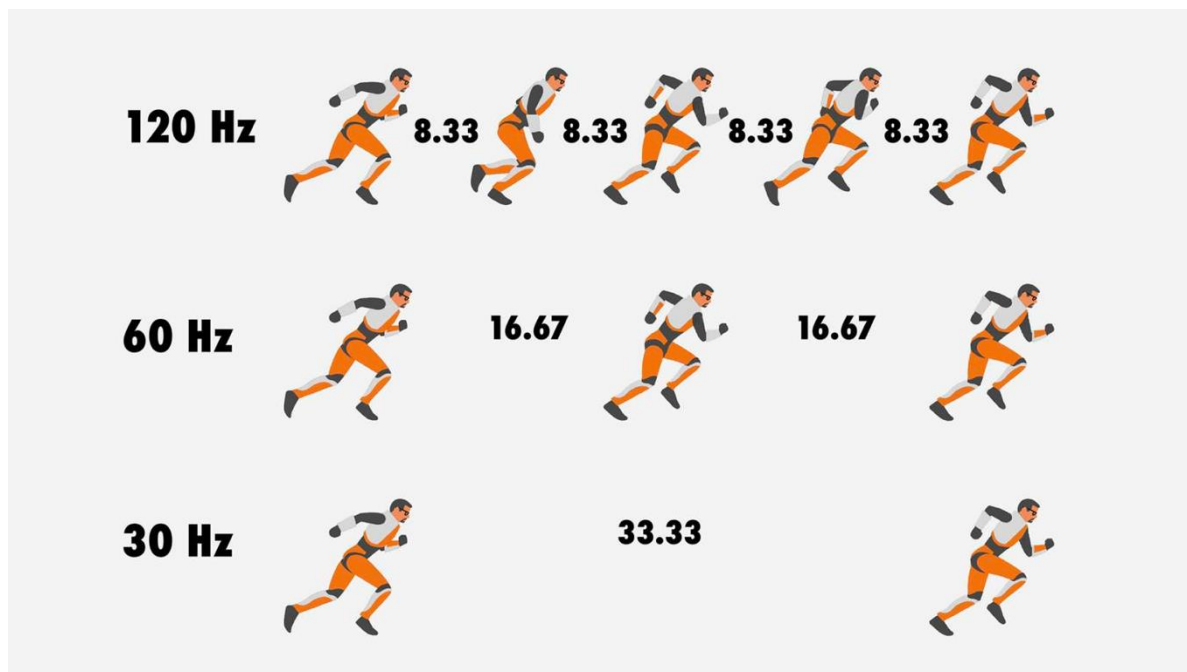


Presenting stimuli



What is refresh rate?

显示器的刷新率是指显示器每秒绘制新图像的次数。其单位为赫兹（Hz）。



Timing stimuli

```
# Wait 3 seconds so people can see the message, then exit gracefully:
```

```
>> core.wait(3)
```

```
# quit the programme
```

```
>> win.close()
```

```
>> core.quit()
```

There are various ways to measure and control timing in PsychoPy®:

- using frame refresh periods (most accurate, least obvious)
- checking the time on **Clock** objects
- using **core.wait()** commands (most obvious, least flexible/accurate)

Timing – Clock()

- Using **core.wait()** is clear and intuitive in your script. But it can't be used while something is changing. For more flexible timing, you could use a **Clock()** object from the core module:

```
#!/usr/bin/env python3
#
# Description:
# core.Clock()
# A convenient class to keep track of time in your experiments.
# You can have as many independent clocks as you like

# import packages
from psychopy import visual, core

# Setup stimulus
win = visual.Window([800, 600])
gabor = visual.GratingStim(win, tex='sin', mask='gauss', sf=5, name='gabor')
gabor.autoDraw = True # Automatically draw every frame
gabor.autoLog = False # Or we'll get many messages about phase change

# Let's draw a stimulus for 5 s, drifting for middle 2.5 s
clock = core.Clock() # set up a clock
while clock.getTime() < 5.0: # Clock times are in seconds
    if 0.5 <= clock.getTime() < 3.0:
        gabor.phase += 0.1 # Increment by 10th of cycle
    win.flip()

core.quit()
win.close()
```

<https://psychopy.org/api/core.html>

```
# initialize a clock
```

```
>> clock = core.Clock()
```

```
# return the current time of your clock
```

```
>> clock.getTime()
```

```
# reset your clock
```

```
>> clock.reset()
```

demo_timing_Clock.py

Timing - frame

The most precise way to control stimulus timing is to present them for a specified number of frames

<https://psychopy.org/coder/codeStimuli.html#timing>

```
#!/usr/bin/env python3
#
# Description:
# The most precise way to control stimulus timing
# is to present them for a specified number of frames.
#
# import modules
from psychopy import visual, core

# Setup stimulus
win = visual.Window([800, 600])
gabor = visual.GratingStim(win, tex='sin', mask='gauss', sf=5,
    name='gabor', autoLog=False)

# Let's draw a stimulus for 200 frames, drifting for frames 50:100
for frameN in range(200): # For exactly 200 frames
    if frameN < 50 or frameN >= 100:
        gabor.draw()
    if 50 <= frameN < 100: # Present stim for a different subset
        gabor.phase += 0.1 # Increment by 10th of cycle
        gabor.draw()
    win.flip()

core.quit()
win.close()
```

If the screen is refreshing at 60 Hz (16.7 ms per frame) and the *getTime()* call reports that the time has reached 1.999 s, then the stimulus will draw again for a frame, in accordance with the *while* loop statement and will ultimately be displayed for 2.0167 s. Alternatively, if the time has reached 2.001 s, there will not be an extra frame drawn.

demo_timing_frames.py

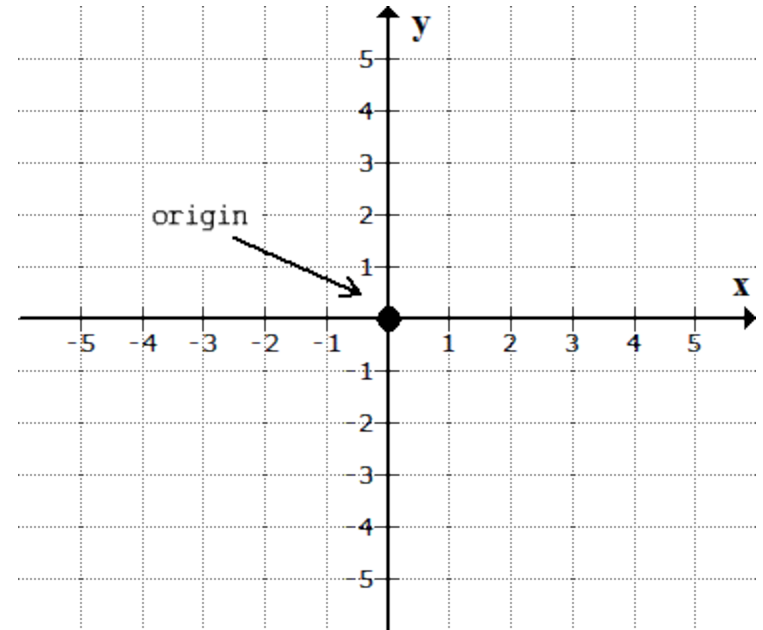
Shapes

<https://psychopy.org/api/visual/index.html>

Psychopy has origin coordinate for screen at the **center**

Shapes (all special classes of **ShapeStim**):

- **ShapeStim** to draw shapes with arbitrary numbers of vertices
- **Rect** to show rectangles
- **Circle** to show circles
- **Polygon** to show polygons
- **Line** to show a line
- **Pie** to show wedges and semi-circles



demo_shapes.py

Collect responses

<https://psychopy.org/api/event.html>

psychopy.event - for keypresses and mouse clicks

`class psychopy.event.Mouse(visible=True, newPos=None, win=None)`

Easy way to track what your mouse is doing.

It needn't be a class, but since Joystick works better as a class this may as well be one too for consistency

Create your *visual.Window* before creating a Mouse.

Parameters:: visible : True or False

makes the mouse invisible if necessary

newPos : None or [x,y]

gives the mouse a particular starting position (pygame *Window* only)

win : None or Window

the window to which this mouse is attached (the first found if None provided)

[\[source\]](#)

`psychopy.event.waitKeys(maxWait=inf, keyList=None, modifiers=False, timeStamped=False, clearEvents=True)`

Same as `~psychopy.event.getKeys`, but halts everything (including drawing) while awaiting input from keyboard.

[\[source\]](#)

Parameters:: maxWait : any numeric value.

Maximum number of seconds period and which keys to wait for. Default is float('inf') which simply waits forever.

keyList : None or []

Allows the user to specify a set of keys to check for. Only keypresses from this set of keys will be removed from the keyboard buffer. If the keyList is *None*, all keys will be checked and the key buffer will be cleared completely. NB, pygame doesn't return timestamps (they are always 0)

modifiers : False or True

If True will return a list of tuples instead of a list of keynames. Each tuple has (keyname, modifiers). The modifiers are a dict of keyboard modifier flags keyed by the modifier name (eg. 'shift' , 'ctrl').

timeStamped : False, True, or Clock

If True will return a list of tuples instead of a list of keynames. Each tuple has (keyname, time). If a *core.Clock* is given then the time will be relative to the *Clock*'s last reset.

clearEvents : True or False

Whether to clear the keyboard event buffer (and discard preceding keypresses) before starting to monitor for new keypresses.

```
>> event.waitKeys(maxWait=5, keyList=['a'])
```

demo_mouse.py

demo_waitKeys.py
demo_getKeys.py

Receive data information

<https://psychopy.org/api/gui.html>

```
# Get subject info with a dialog
participant = {'Participant ID': '', 'Age': '', 'Gender': ''}
dlg = gui.DlgFromDict(participant, title='Dialogue Box Exercise', sortKeys=False)
if dlg.OK == False:
    core.quit() # user pressed cancel button
```

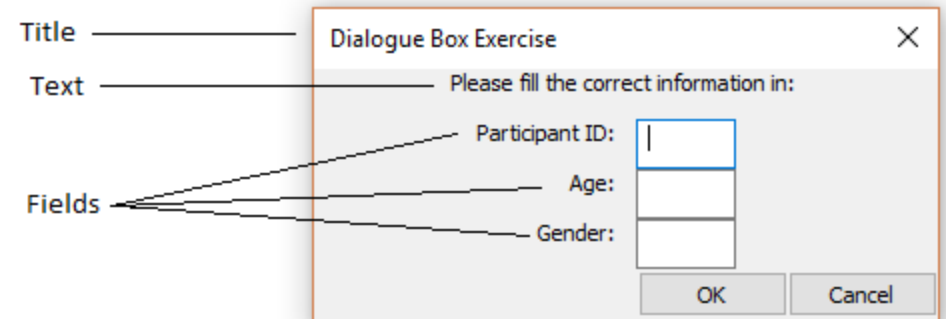
psychopy.gui - create dialogue boxes

DlgFromDict

`class psychopy.gui.DlgFromDict(dictionary, title='', fixed=None, order=None, tip=None, screen=-1, sortKeys=True, copyDict=False, labels=None, show=True, sort_keys=None, copy_dict=None)` [\[source\]](#)

Creates a dialogue box that represents a dictionary of values. Any values changed by the user are change (in-place) by this dialogue box.

- Parameters::**
- **dictionary** (*dict*) – A dictionary defining the input fields (keys) and pre-filled values (values) for the user dialog
 - **title** (*str*) – The title of the dialog window
 - **labels** (*dict*) – A dictionary defining labels (values) to be displayed instead of key strings (keys) defined in *dictionary*. Not all keys in *dictionary* need to be contained in labels.
 - **fixed** (*list*) – A list of keys for which the values shall be displayed in non-editable fields
 - **order** (*list*) – A list of keys defining the display order of keys in *dictionary*. If not all keys in *dictionary* are contained in *order*, those will appear in random order after all ordered keys.
 - **tip** (*list*) – A dictionary assigning tooltips to the keys
 - **screen** (*int*) – Screen number where the Dialog is displayed. If -1, the Dialog will be displayed on the primary screen.
 - **sortKeys** (*bool*) – A boolean flag indicating that keys are to be sorted alphabetically.



demo_dialogueBox.py

Global Event Keys

<https://psychopy.org/coder/globalKeys.html>

```
# Define the shutdown function
def quit_func():
    text_msg.text = u"你已按下q或ESC键！ "
    text_msg.draw()
    win.flip()
    core.wait(3)
    win.close()
    core.quit()

# Add global keys so that we can shutdown at any time
for key in ['q', 'escape']:
    event.globalKeys.add(key, func=quit_func)
```

Global Event Keys

Global event keys are single keys (or combinations of a single key and one or more “modifier” keys such as Ctrl, Alt, etc.) with an associated Python callback function. This function will be executed if the key (or key/modifiers combination) was pressed.

Note

Global event keys only work with the *pyglet* backend, which is the default.

PsychoPy® fully automatically monitors and processes key presses during most portions of the experimental run, for example during *core.wait()* periods, or when calling *win.flip()*. If a global event key press is detected, the specified function will be run immediately. You are not required to manually poll and check for key presses. This can be particularly useful to implement a global “shutdown” key, or to trigger laboratory equipment on a key press when testing your experimental script – without cluttering the code. But of course the application is not limited to these two scenarios. In fact, you can associate any Python function with a global event key.

All active global event keys are stored in *event.globalKeys*.

Adding a global event key (simple)

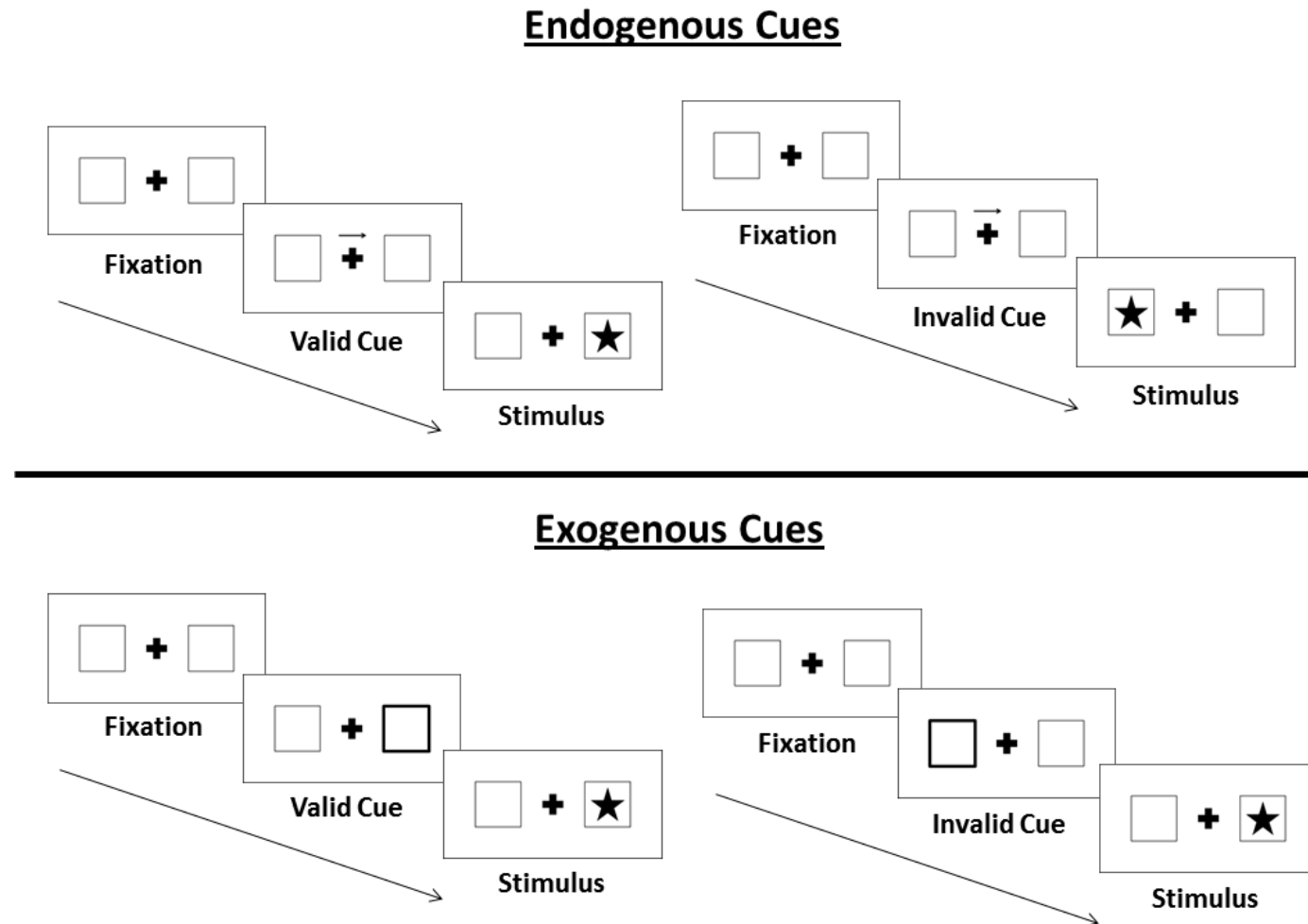
First, let’s ensure no global event keys are currently set by calling `func: event.globalKeys.clear()`.

```
>>> from psychopy import event
>>> event.globalKeys.clear()
```

demo_globalKeys.py

Putting it all together

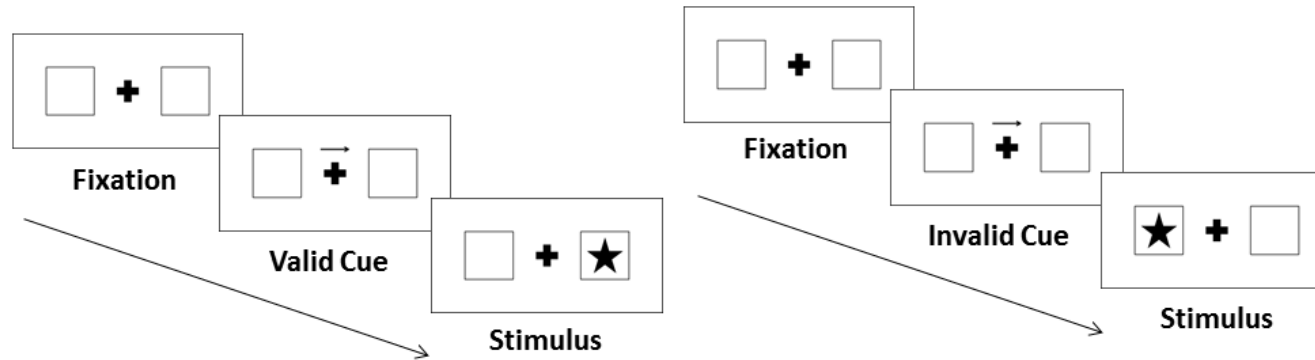
Posner Cueing Paradigm (Posner, 1980)



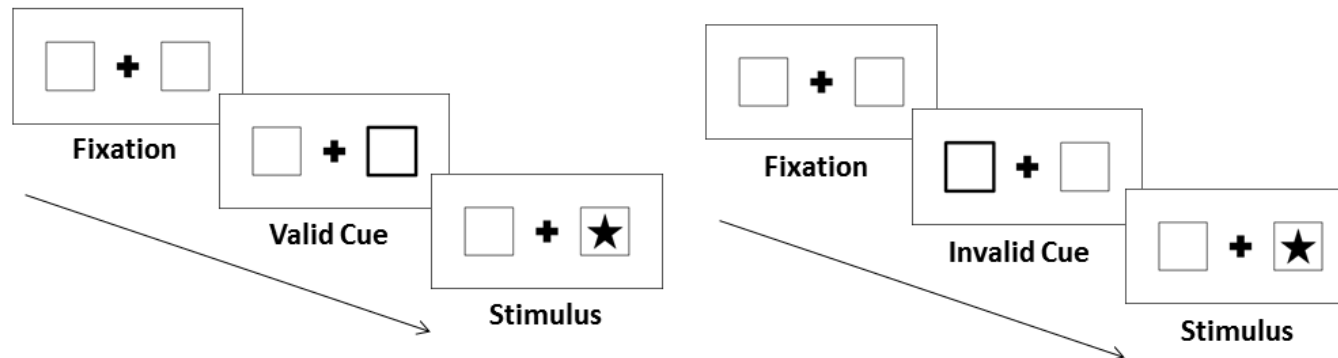
More elegant script

Posner Cueing Paradigm (Posner, 1980)

Endogenous Cues



Exogenous Cues



Hands-on task

修改你的代码，把它变成Exogenous task。