```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
This experiment was created using PsychoPy3 Experiment Builder (v3.2.4),
    on Thu Feb 13 09:46:41 2020
If you publish work using this script the most relevant publication is:

    Peirce J, Gray JR, Simpson S, MacAskill M, Höchenberger R, Sogo H, Kastman E, Lindeløv JK. (2019)
        PsychoPy2: Experiments in behavior made easy Behav Res 51: 195.
        https://doi.org/10.3758/s13428-018-01193-y

"""

from __future__ import absolute_import, division

from psychopy import locale_setup
from psychopy import prefs
from psychopy import sound, gui, visual, core, data, event, logging, clock
from psychopy.constants import (NOT_STARTED, STARTED, PLAYING, PAUSED,
                                STOPPED, FINISHED, PRESSED, RELEASED, FOREVER)

import numpy as np  # whole numpy lib is available, prepend 'np.'
from numpy import (sin, cos, tan, log, log10, pi, average,
                   sqrt, std, deg2rad, rad2deg, linspace, asarray)
from numpy.random import random, randint, normal, shuffle
import os  # handy system and path functions
import sys  # to get file system encoding

from psychopy.hardware import keyboard

# Ensure that relative paths start from the same directory as this script
_thisDir = os.path.dirname(os.path.abspath(__file__))
os.chdir(_thisDir)

# Store info about the experiment session
psychopyVersion = '3.2.4'
expName = 'demo'  # from the Builder filename that created this script
expInfo = {'participant': '', 'session': '001'}
dlg = gui.DlgFromDict(dictionary=expInfo, sortKeys=False, title=expName)
if dlg.OK == False:
    core.quit()  # user pressed cancel
expInfo['date'] = data.getDateStr()  # add a simple timestamp
expInfo['expName'] = expName
expInfo['psychopyVersion'] = psychopyVersion

# Data file name stem = absolute path + name; later add .psyexp, .csv, .log, etc
filename = _thisDir + os.sep + u'data/%s_%s_%s' % (expInfo['participant'], expName, expInfo['date'])

# An ExperimentHandler isn't essential but helps with data saving
thisExp = data.ExperimentHandler(name=expName, version='',
    extraInfo=expInfo, runtimeInfo=None,
    originPath='/Users/yangzhihan/Desktop/projects/perception_lab/demo_program/demo.py',
    savePickle=True, saveWideText=True,
    dataFileName=filename)
# save a log file for detail verbose info
logFile = logging.LogFile(filename+'.log', level=logging.EXP)
logging.console.setLevel(logging.WARNING)  # this outputs to the screen, not a file

endExpNow = False  # flag for 'escape' or other condition => quit the exp
frameTolerance = 0.001  # how close to onset before 'same' frame

# Start Code - component code to be run before the window creation

# Setup the Window
win = visual.Window(
```

```
    size=(1024, 768), fullscr=True, screen=0,
    winType='pyglet', allowGUI=False, allowStencil=False,
    monitor='testMonitor', color=[0,0,0], colorSpace='rgb',
    blendMode='avg', useFBO=True,
    units='height')
# store frame rate of monitor if we can measure it
expInfo['frameRate'] = win.getActualFrameRate()
if expInfo['frameRate'] != None:
    frameDur = 1.0 / round(expInfo['frameRate'])
else:
    frameDur = 1.0 / 60.0  # could not measure, so guess

# create a default keyboard (e.g. to check for escape)
defaultKeyboard = keyboard.Keyboard()

# Initialize components for Routine "instructions"
instructionsClock = core.Clock()
instructions_text = visual.TextStim(win=win, name='instructions_text',
    text='Welcome!',
    font='Arial',
    pos=(0, 0), height=0.1, wrapWidth=None, ori=0,
    color='white', colorSpace='rgb', opacity=1,
    languageStyle='LTR',
    depth=0.0);
instructions_keyresp = keyboard.Keyboard()

# Initialize components for Routine "show_condition"
show_conditionClock = core.Clock()
show_condition_text = visual.TextStim(win=win, name='show_condition_text',
    text='default text',
    font='Arial',
    pos=(0, 0), height=0.1, wrapWidth=None, ori=0,
    color='white', colorSpace='rgb', opacity=1,
    languageStyle='LTR',
    depth=0.0);

# Create some handy timers
globalClock = core.Clock()  # to track the time since experiment started
routineTimer = core.CountdownTimer()  # to track time remaining of each (non-slip) routine

# ------Prepare to start Routine "instructions"-------
# update component parameters for each repeat
instructions_keyresp.keys = []
instructions_keyresp.rt = []
# keep track of which components have finished
instructionsComponents = [instructions_text, instructions_keyresp]
for thisComponent in instructionsComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
instructionsClock.reset(-_timeToFirstFrame)  # t0 is time of first possible flip
frameN = -1
continueRoutine = True

# -------Run Routine "instructions"-------
while continueRoutine:
    # get current time
    t = instructionsClock.getTime()
    tThisFlip = win.getFutureFlipTime(clock=instructionsClock)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
```

component

component 2

initialize all components

these attributes are added not assigned

2 styles of timers
one using conventional timer
the other using screen refresh

```
    # update/draw components on each frame

    # *instructions_text* updates
    if instructions_text.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
        # keep track of start time/frame for later
        instructions_text.frameNStart = frameN  # exact frame index
        instructions_text.tStart = t  # local t and not account for scr refresh
        instructions_text.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(instructions_text, 'tStartRefresh')  # time at next scr refresh
        instructions_text.setAutoDraw(True)

    # *instructions_keyresp* updates
    waitOnFlip = False
    if instructions_keyresp.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
        # keep track of start time/frame for later
        instructions_keyresp.frameNStart = frameN  # exact frame index
        instructions_keyresp.tStart = t  # local t and not account for scr refresh
        instructions_keyresp.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(instructions_keyresp, 'tStartRefresh')  # time at next scr refresh
        instructions_keyresp.status = STARTED
        # keyboard checking is just starting
        waitOnFlip = True
        win.callOnFlip(instructions_keyresp.clock.reset)  # t=0 on next screen flip
        win.callOnFlip(instructions_keyresp.clearEvents, eventType='keyboard')  # clear events on next screen flip
    if instructions_keyresp.status == STARTED and not waitOnFlip:
        theseKeys = instructions_keyresp.getKeys(keyList=['space'], waitRelease=False)
        if len(theseKeys):
            theseKeys = theseKeys[0]  # at least one key was pressed

            # check for quit:
            if "escape" == theseKeys:
                endExpNow = True
            instructions_keyresp.keys = theseKeys.name  # just the last key pressed
            instructions_keyresp.rt = theseKeys.rt
            # a response ends the routine
            continueRoutine = False

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in instructionsComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# -------Ending Routine "instructions"-------
for thisComponent in instructionsComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
thisExp.addData('instructions_text.started', instructions_text.tStartRefresh)
thisExp.addData('instructions_text.stopped', instructions_text.tStopRefresh)
# check responses
if instructions_keyresp.keys in ['', [], None]:  # No response was made
    instructions_keyresp.keys = None
thisExp.addData('instructions_keyresp.keys',instructions_keyresp.keys)
if instructions_keyresp.keys != None:  # we had a response
    thisExp.addData('instructions_keyresp.rt', instructions_keyresp.rt)
```

why set twice? timeOnFlip is more accurate

Determines whether the stimulus should be automatically drawn on every frame flip.

process keyboard inputs

decide whether this Routine should end

record data

```
thisExp.addData('instructions_keyresp.started', instructions_keyresp.tStartRefresh)
thisExp.addData('instructions_keyresp.stopped', instructions_keyresp.tStopRefresh)
thisExp.nextEntry()
# the Routine "instructions" was not non-slip safe, so reset the non-slip timer
routineTimer.reset()

# set up handler to look after randomisation of conditions etc
trials = data.TrialHandler(nReps=1, method='random',
    extraInfo=expInfo, originPath=-1,
    trialList=data.importConditions('csvs/noise_conditions.csv'),
    seed=None, name='trials')
thisExp.addLoop(trials)  # add the loop to the experiment
thisTrial = trials.trialList[0]  # so we can initialise stimuli with some values
# abbreviate parameter names if possible (e.g. rgb = thisTrial.rgb)
if thisTrial != None:
    for paramName in thisTrial:
        exec('{} = thisTrial[paramName]'.format(paramName))

for thisTrial in trials:
    currentLoop = trials
    # abbreviate parameter names if possible (e.g. rgb = thisTrial.rgb)
    if thisTrial != None:
        for paramName in thisTrial:
            exec('{} = thisTrial[paramName]'.format(paramName))

    # ------Prepare to start Routine "show_condition"-------
    routineTimer.add(1.000000)
    # update component parameters for each repeat
    show_condition_text.setText(condition)
    # keep track of which components have finished
    show_conditionComponents = [show_condition_text]
    for thisComponent in show_conditionComponents:
        thisComponent.tStart = None
        thisComponent.tStop = None
        thisComponent.tStartRefresh = None
        thisComponent.tStopRefresh = None
        if hasattr(thisComponent, 'status'):
            thisComponent.status = NOT_STARTED
    # reset timers
    t = 0
    _timeToFirstFrame = win.getFutureFlipTime(clock="now")
    show_conditionClock.reset(-_timeToFirstFrame)  # t0 is time of first possible flip
    frameN = -1
    continueRoutine = True

    # -------Run Routine "show_condition"-------
    while continueRoutine and routineTimer.getTime() > 0:
        # get current time
        t = show_conditionClock.getTime()
        tThisFlip = win.getFutureFlipTime(clock=show_conditionClock)
        tThisFlipGlobal = win.getFutureFlipTime(clock=None)
        frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
        # update/draw components on each frame

        # *show_condition_text* updates
        if show_condition_text.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
            # keep track of start time/frame for later
            show_condition_text.frameNStart = frameN  # exact frame index
            show_condition_text.tStart = t  # local t and not account for scr refresh
            show_condition_text.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(show_condition_text, 'tStartRefresh')  # time at next scr refresh
            show_condition_text.setAutoDraw(True)
        if show_condition_text.status == STARTED:
            # is it time to stop? (based on global clock, using actual start)
            if tThisFlipGlobal > show_condition_text.tStartRefresh + 1-frameTolerance:
                # keep track of stop time/frame for later
```

loop through all the trials

this loop is forced stopped by this

not used

```
266             show_condition_text.tStop = t  # not accounting for scr refresh
267             show_condition_text.frameNStop = frameN  # exact frame index
268             win.timeOnFlip(show_condition_text, 'tStopRefresh')  # time at next scr refresh
269             show_condition_text.setAutoDraw(False)
270
271         # check for quit (typically the Esc key)
272         if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
273             core.quit()
274
275         # check if all components have finished
276         if not continueRoutine:  # a component has requested a forced-end of Routine
277             break
278         continueRoutine = False  # will revert to True if at least one component still running
279         for thisComponent in show_conditionComponents:
280             if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
281                 continueRoutine = True
282                 break  # at least one component has not yet finished
283
284         # refresh the screen
285         if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
286             win.flip()
287
288     # -------Ending Routine "show_condition"-------
289     for thisComponent in show_conditionComponents:
290         if hasattr(thisComponent, "setAutoDraw"):
291             thisComponent.setAutoDraw(False)
292     trials.addData('show_condition_text.started', show_condition_text.tStartRefresh)
293     trials.addData('show_condition_text.stopped', show_condition_text.tStopRefresh)
294     thisExp.nextEntry()
295
296 # completed 1 repeats of 'trials'
297
298
299 # Flip one final time so any remaining win.callOnFlip()
300 # and win.timeOnFlip() tasks get executed before quitting
301 win.flip()
302
303 # these shouldn't be strictly necessary (should auto-save)
304 thisExp.saveAsWideText(filename+'.csv')
305 thisExp.saveAsPickle(filename)
306 logging.flush()
307 # make sure everything is closed down
308 thisExp.abort()  # or data files will save again on exit
309 win.close()
    core.quit()
```

update screen

go over all components
if they had drawn anything
stop them from drawing anymore!

stop all components from drawing

record start and stop

proceed to next entry