

第五章 Psychopy 入门技术

5.1 Psychopy 简介

前言部分已经介绍过，有多个基于 Python 的工具包可以用于编写认知神经科学实验任务，Psychopy 就是近期非常流行的一套实验编程工具包。依托于 Python，Psychopy 支持多个主要的操作系统，包括 Windows, Linux 和 Mac。该软件包的作者为 Nottingham 大学的 Jon Peirce 博士。根据 Psychopy 官网的统计，最近 5 年 Psychopy 的用户数量急剧增加。保守估计，Psychopy 的用户数量目前为 12000 左右。

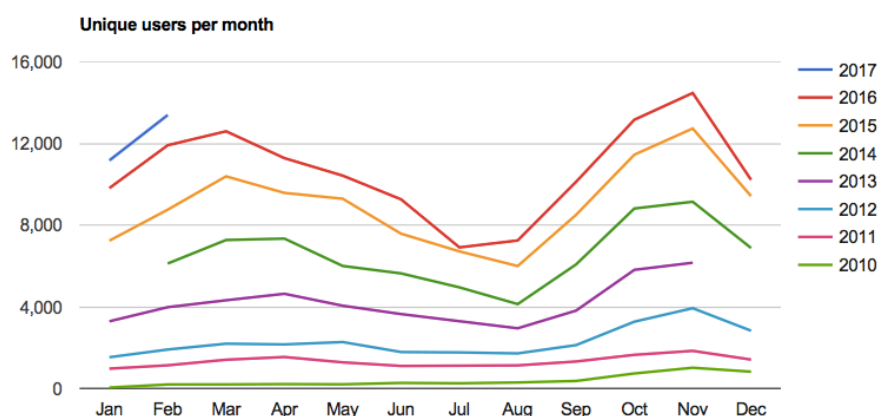


图 5.1 Psychopy 使用统计。这里显示的数据更多地代表新用户 / 初级用户的数量，多数用户把 Psychopy 作为 Python 的扩展包使用，因此使用 Psychopy 时并不会自动反馈使用情况。来源：

<http://www.psychopy.org/usage.php>

Psychopy 采用基于对象的编程技术，功能强大。目前，Psychopy 已经实现的功能包括：复杂视觉刺激的生成和呈现、键盘等反应设备的控制，以及与眼动、脑电和核磁等设备的整合。此外，Psychopy 还提供有部分便捷的实验工具，比如颜色系统的转换、显示器的校准等。Psychopy 对视觉刺激和反应设备的控制实际上依赖于两个 Python 的多媒体扩展包，即 Pygame 和 Pyglet（新近版本不再依赖 Pygame）。前者基于 SDL，而后者则基于 OpenGL。前面章节已经详细介绍了 Pygame，Psychopy 是对 Pygame 和 Pyglet 的功能封装，使用 Pygame 和 Pyglet 也能够实现复杂的实验任务。

Psychopy 对于眼动设备的控制主要依赖于厂商提供的 API，比如，控制 Eyelink 眼动仪还需要安装 SR Research 提供的 Python 扩展包 Pylink。Psychopy 与脑电、Arduino

等设备的整合则主要依赖于并口和串口通讯方面的工具包。需要指出的是，这些工具都是通用的，并不依赖于 Psychopy。设置好 Python 编程环境后，即使不安装 Psychopy 也能使用并实现与这些实验设备的交互。

Psychopy 的优点是功能丰富，缺点是依赖的 Python 扩展包很多，手动安装容易出现软件包依赖关系方面的问题。Psychopy 提供有独立安装包（standalone 版本），该版本打包有 Python 编译器，可以作为单独的应用程序来使用。但是这种安装方法会导致用户丧失使用 Psychopy 的灵活性，比如，即使用户使用其他编辑器来写代码，他们仍然需要使用 Psychopy 的 Coder 界面打开这些代码后才能运行。如果用户自己手动安装 Psychopy 所依赖的扩展包，部分扩展包需要编译，用户可能会遇到困难。但是，我本人推荐手动安装的方式，这可以让我们轻松地使用操作系统默认的 Python 编译器来运行实验代码。

Psychopy 的低版本的部分功能在高版本中被放弃或者重新实现，因此在版本兼容方面存在问题。在使用 Psychopy 的过程中，如果遇到问题可以参考 Psychopy 的在线文档（<http://www.psychopy.org/documentation.html>）。不过，由于 Psychopy 的更新速度很快，部分新功能在文档中可能并没有体现。但是瑕不掩瑜，Psychopy 能够满足多数认知神经科学领域的实验需要，本章将简要介绍 Psychopy 的主要功能。在第 6 章，我们会通过一个完整的 psychopy 实验代码来介绍如何高效地使用 Psychopy 编制实验程序。

5.2 安装 Psychopy

上文已经提及，Psychopy 可以作为一个单独的应用程序来使用，也可以作为一个 Python 的扩展包来使用。对于初学者，建议先尝试 Psychopy 的独立安装包

（standalone 版本），对 Python 比较熟悉以后可以尝试手动安装的方法，把 Psychopy 安装到操作系统的默认 Python 发行版中。

5.2.1 使用独立安装包安装 Psychopy

Psychopy 目前的稳定版本是 1.84.2。Windows（所有版本）用户请下载 StandalonePsychoPy-1.84.2-win32.exe，Mac OSX 用户请下载 StandalonePsychoPy-1.84.2-OSX_64bit.dmg。需要指出的是，Mac OSX 版本的 Psychopy 打包了 64 位的 Python，因此运行 32 位 Python 编写的程序时可能会遇到一些问题。Psychopy 没有 Linux 版本的独立安装包，如果使用 Ubuntu（或其他 Debian 衍生的 Linux 发行版），联网并运行

下述命令即可安装 Psychopy 及其依赖的所有 Python 扩展包。但是这种方法安装的 psychopy 版本通常会较低。

```
sudo apt-get install psychopy
```

5.2.2 手动安装 Psychopy 及其依赖的 Python 扩展包

使用 pip 可以方便地安装各种 Python 的扩展包。但是 Psychopy 依赖的部分扩展包不支持 pip 安装，用户需要下载后手动安装，这些包主要包括 wxPython 和 pyo。详情见 Psychopy 的官网有最新的安装指南

(<http://www.psychopy.org/installation.html#manual-install>)。

wxPython: <https://wxpython.org/download.php>

pyo: <http://ajaxsoundstudio.com/software/pyo/>

安装完上面两个扩展包以后便可以打开终端执行的下面这条命令来安装 Psychopy 依赖的扩展包，以及 Psychopy 本身。在 Windows 操作系统下，在开始菜单搜索“cmd”，然后按回车键（ENTER）打开终端。在执行下述命令前，用户可能需要首先把 pip 升级到最新版本（“pip install update pip”）。如果是 Mac OSX 操作系统，可能需要管理员权限，在下述命令前加“sudo”。

```
pip install numpy scipy matplotlib pandas pyopengl pygame pillow moviepy lxml openpyxl  
xlrd configobj pyyaml gevent greenlet msgpack-python psutil tables requests[security]  
pysf cffi pysoundcard pysoundfile seaborn psychopy_ext python-bidi psychopy
```

Psychopy 有 Coder 和 Builder 两个用户界面。前者是一个编辑器，而后者是一个支持拖拉图标创建实验程序的图形界面。任何采用拖拉图标方式编程的方法（比如 E-Prime）在灵活性方面都很有限，因此本书不会涉及 Builder。鉴于手动安装 Psychopy 的难度，为了便于初学者接受，本书的示例将使用 Psychopy 的 Coder 编辑器。但是，所有代码均不依赖于 Psychopy 的图形界面，用户也可以使用其他 Python 编辑器来编辑和运行这些示例代码。

5.3 屏幕和视觉刺激的度量单位

5.3.1 视角

在视觉实验中，我们关心的不是刺激本身的物理尺寸，而是其在视网膜的投影的大小。同样一个刺激，距离眼睛更远时，它在视网膜上的投影就越小。因此我们通常使用“视角”（visual angle）来描述和控制视觉刺激的大小。在显示设备呈现刺激时，这些设备度量刺激大小的单位是像素（pixel）。因此，在编写实验程序呈现刺激时我们需要以“像素”单位，但是在设计实验和报告结果时我们需要以“视角”为单位。这里就涉及视角和像素的单位转换。实现两者粗略的转换需要以下几个参数：1) 显示设备的可视区域的物理尺寸（cm）；2) 显示设备的可视区域的大小（像素），以及显示设备与眼睛的距离（cm）。像素和视角的间的转化通常有两种简便的方法，即反正切法和“弧度”法。具体的计算方法见图 5.2。

在 Psychopy 中可以使用像素、视角、厘米等单位来设定刺激的大小。需要注意的是，如果使用视角为刺激的单位，那么需要用户首先创建一个 Monitor 对象来告知 Psychopy 显示设备的物理尺寸(cm)，视距(cm)，以及显示设备以像素为单位的大小。这是新用户经常忘记的一件事情。

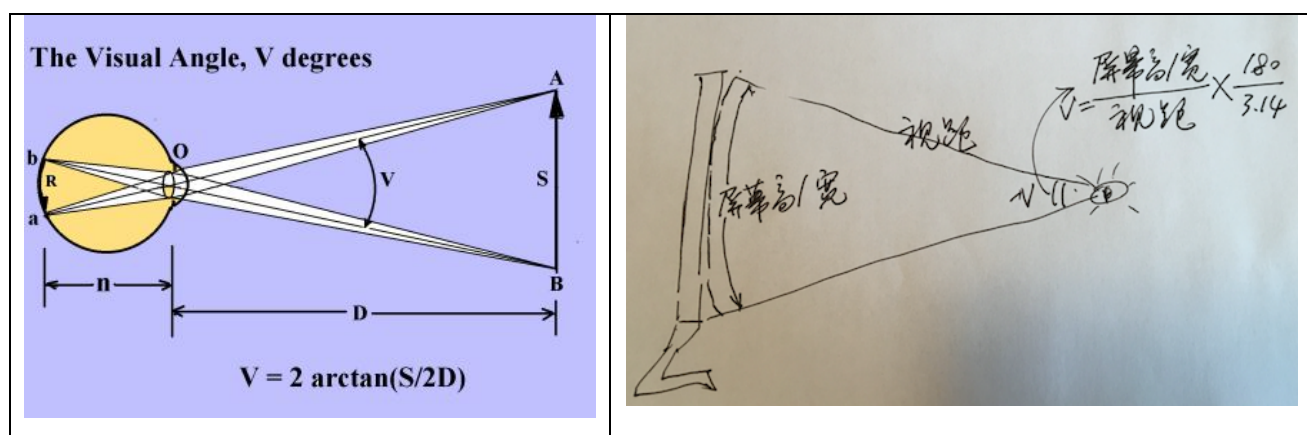


图 5.2 采用反正切和弧度法近似计算视角。

5.3.2 Psychopy 所特有的刺激度量单位

除了视角、厘米、像素，Psychopy 还支持两种特殊的单位，“高度”（height）和“标准单位”（normalized）。对于“高度”单位来说，显示设备的可视区域的高度被设为 1。对于 Psychopy 来说，屏幕的中心的坐标为 (0, 0)。因此使用高度单位时，对于 16:10

的屏幕来讲，左下角的坐标为 $(-0.8, -0.5)$ ，右上角的坐标为 $(0.8, 0.5)$ 。对于“标准”单位，屏幕的 X 和 Y 坐标范围分别为 $(-1, 1)$ 。这两种刺激单位的特点是刺激的大小会随显示设备的大小自动缩放，使用时不需要提供视距，以及显示设备的物理大小等信息。

5.3.3 显示设备的坐标系

前面介绍过，在 Pygame 中屏幕的左上角的坐标是 $(0, 0)$ ，右下角的坐标为屏幕支持的最大分辨率减去 1 个像素。在 Psychopy 中，屏幕的坐标的原点 $(0, 0)$ 则是屏幕中心，和大家熟悉的几何参照系相同。

5.4 颜色编码

颜色是一种主观的感受。编码颜色可以有多种不同的方法和标准 (color spaces)，比如 RGB, DKL 和 HSL 等。这里需要指出的是，虽然 DKL 编码便于用户选取等亮度 (equi-luminant) 的颜色，但是受显示设备本身的色域 (color gamut) 的限制，并不是所有 DKL 编码的颜色都能够在显示设备上呈现出来。颜色是一个复杂的问题，具体的讨论超出了本书的范围，感兴趣的读者可以参考下面这本书。这里需要介绍的几种 Psychopy 接受的颜色编码。

Westland, S., Ripamonti, C., & Cheung, V. (2004). *Computational Colour Science Using MATLAB: A Practical Approach*. Hoboken, NJ: Wiley-Blackwell.

- X11 颜色名。比如“black”和“lightpink”。其他 X11 标准颜色名可以从该链接获取，<http://cng.seas.rochester.edu/CNG/docs/x11color.html>。
- RGB 颜色。这里与大家熟知的 RGB 编码不同，RGB 三个通道的取值范围是 -1 到 +1，而不是 0 到 255。因此，RGB 颜色 $(0, 0, 0)$ 对应的编码是 $[-1, -1, -1]$ 或者 $(-1, -1, -1)$ 。这样视觉实验中常用的中间灰就可以表示为 $[0, 0, 0]$ 。
- 使用三个 16 进制数表示 RGB 颜色，比如白色表示为“#FFFFFF”。在 Psychopy 中使用时需要以字符串形式表示，和使用 X11 颜色名的方法相同。
- DKL 和 HSL 编码不常用，需要的读者参考该链接，<http://www.psychopy.org/general/colours.html>。

上面提及，虽然颜色有各种不同的颜色空间，但是显示设备的色域有限，仅能呈现颜色空间中的部分颜色。RGB 颜色编码与显示设备关联最密切，建议在 Psychopy 中使用该编码方法。如需使用 DKL 编码，建议首先使用 Psychopy 提供了颜色编码转化的工具将颜色转换成 RGB 格式。

5.5 Psychopy 实例

在这个部分，我们使用一个示例来简要介绍一下 Psychopy 的一些常用功能模块。在这个示例中，我们使用 Psychopy 的 Coder 界面（见图 5.3）。该界面上方的窗口为编辑器，下方的窗口显示代码执行的结果，并包含一个 Python 终端，以便于用户调试代码。如果不需要，可以隐藏下方的窗口（菜单项 View -> Show Output/Shell）。

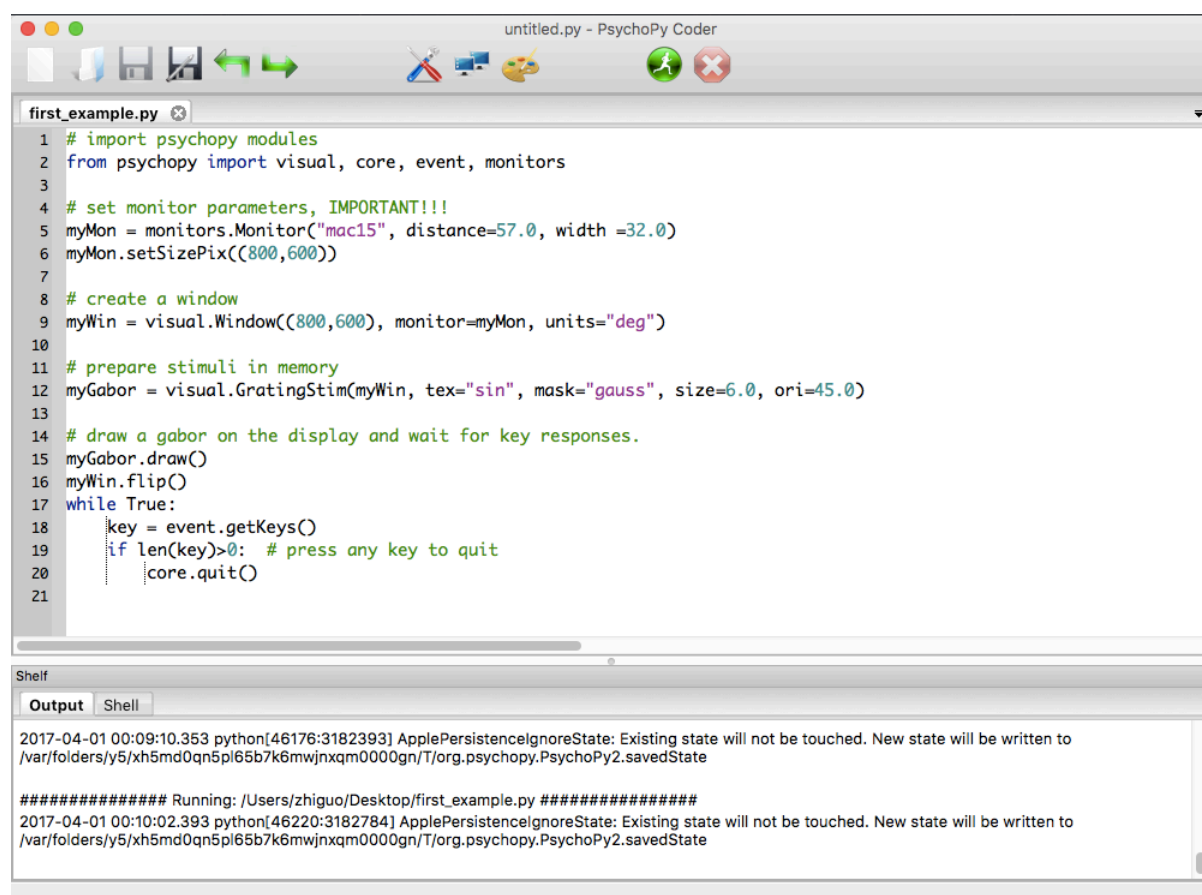


图 5.3 Psychopy 的 Coder 界面及示例代码。

上述代码需要首先保存（扩展名为.py）才能运行（点击“运行”按钮）。首先要指出的是，在前面章节已经介绍过，写代码时要加上清晰明确的注释，以便于后续的维护和修改。注释需要以“#”号开始，可以单起一行，也可以放在一行代码的后面。下面介绍这段代码的各个部分的功能。

- 第 2 行：导入 Psychopy 中的 visual, core, event 和 monitor 模块（module）。
- 第 4-5 行：创建一个 monitors 示例。这一步非常重要。Psychopy 支持多种刺激大小的单位，比如使用“deg”（视角）的话，我们需要知道显示器的物理尺寸，视距，显示器的分辨率（比如，1024 x 768 像素）。因此，在代码起始的地方我们需要首先创建一个 monitor 实例。这里我们把显示器命名为“mac15”。如果再添加一句“myMon.save()”，Psychopy 便会保存一个名为“mac15”的文本文件。再次使用时，直接调用该文本文件即可。
- 第 9 行：这里创建了一个窗口（Window），使用我们创建的 monitor 实例（myMon），同时设定刺激的单位是视角。
- 第 12 行：创建一个 gabor。这里我们需要设定在窗口 myWin 中呈现该刺激。这时 gabor 只是在内存中，如果需要让它显示在屏幕上，我们需要调用它的一个方法 [draw()；第 15 行]，同时 flip 窗口 myWin（第 16 行）。
- 第 17-20 行：使用一个 while 循环等待按键反应。这里我们使用 event 模块的 getKeys() 函数来检测按键。如果有任一键被按下，该函数返回的 list 的长度会大于 0，这时我们就调用 core 模块的 quit() 方法退出 Psychopy。

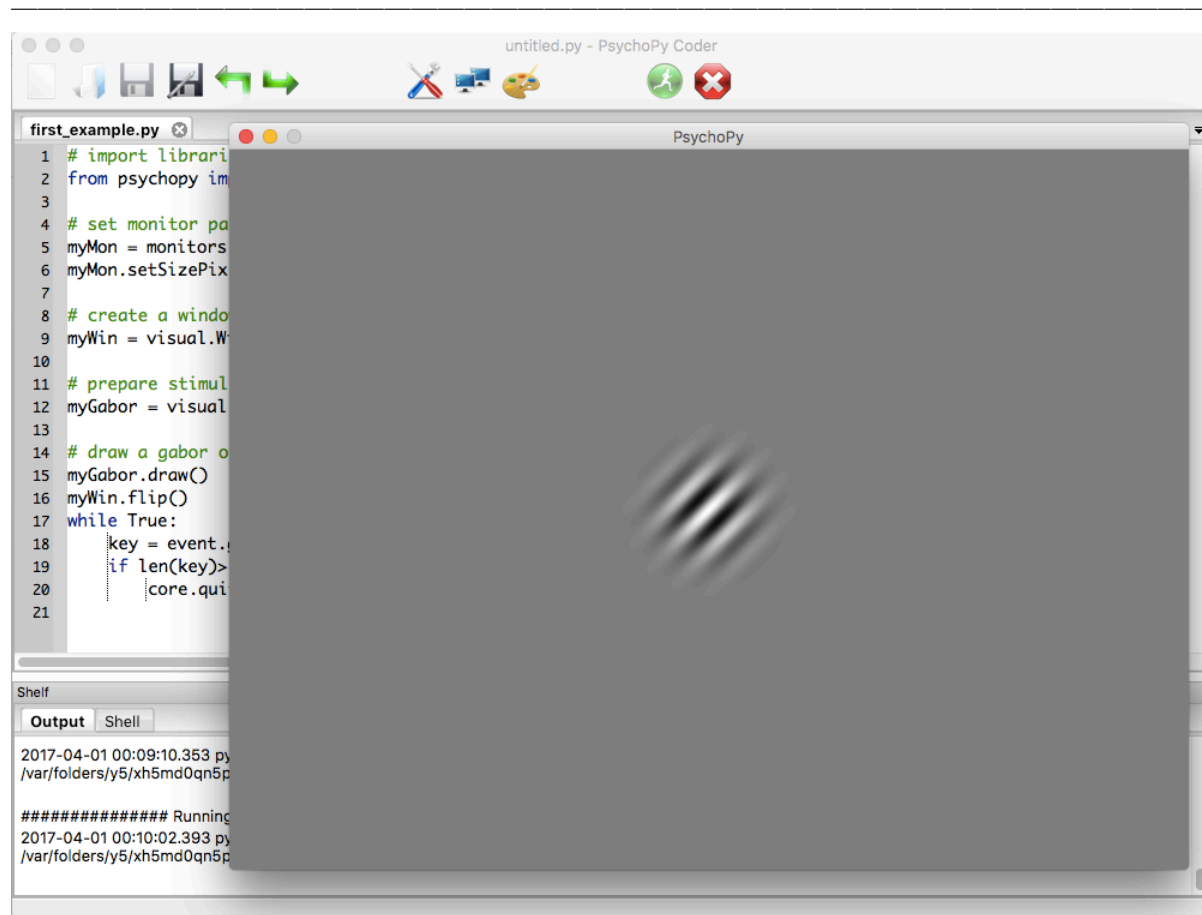


图 5.4 示例代码 1 的运行结果

上述示例代码的运行结果见图 5.4。在该实例代码使用了我们常用到的几个 Psychopy 功能模块。其中最重要的模块是 visual 和 event。这两个模块涉及到视觉刺激的呈现和反应的收集，特别是 visual 模块是 Psychopy 最成熟的功能模块。该模块封装了多种常用的复杂视觉刺激，比如 gratings/gabor 和 checker board 等。这时作者使用 Psychopy 的主要原因，其他功能模块与 Pygame 相比并无特别的优点。Psychopy 也有一些实验控制和数据保存方面的模块，作者认为初学者可能不易接受，因此在示例代码中将采用与前面 Pygame 相关章节相同的方法。

在我们具体介绍各个主要模块前，需要重点指出的是 Psychopy 一个关键优势是它支持“动态”刺激控制。比如，在上述示例代码中，添加下图中高亮的那行代码就可以让 gabor 匀速旋转（屏幕每刷新一次方向增加 0.1 度）。采用该方法，我们还可以动态改变刺激的对比度、颜色、位置、大小等属性。


```
13
14 # draw a gabor on the display and wait for key responses.
15 while True:
16     myGabor.setOri(0.1, "+")
17     myGabor.draw()
18     myWin.flip()
19     key = event.getKeys()
20     if len(key)>0: # press any key to quit
21         core.quit()
```

5.6 Visual 模块

Visual 模块的主要功能是控制各种刺激在显示设备上的呈现。这里我们介绍四个重要的次级模块：Window, gratingStim, RadialStim 和 aperture。

5.6.1 Window

要呈现刺激首先需要创建一个窗口。Psychopy 支持多个窗口。上文已经提及，在磁共振实验中我们可以用一个窗口呈现视觉刺激给被试，另一个窗口呈现任务的参数个进程给主试。创建窗口是需要提供多个参数，关键的几个参数包括 size、color、fullscr（是否全屏模式）、monitor、units（刺激单位）、gamma（显示器 gamma 值）、screen（窗口编号）、allowStencil（与 aperture 功能相关）。

```
visual.Window(size=(800, 600), pos=None, color=(0, 0, 0), colorSpace='rgb', rgb=None,
dki=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', checkTiming=True,
useFBO=False, useRetina=False, autoLog=True)
```

图 5.5 给出一段示例代码来介绍 Window 模块的主要参数和方法。这里需要强调的是 Window 模块支持“抓屏”操作（第 9-10 行），这在部分实验中可能很有用。比如，在 Eyelink 眼动实验是，我们就可以抓屏并传输到 Eyelink 眼动仪主机，这样主试就可以监控被试任务的进程。此外，Window 还支持输出帧率（fps()）以及保存屏幕刷新间隔（saveFrameIntervals()）等功能。

需要强调的是，在视知觉实验中需要首先校准显示器。在创建窗口是需要提供显示器的 gamma 值。显示器的 gamma 校正在后续章节中会介绍。

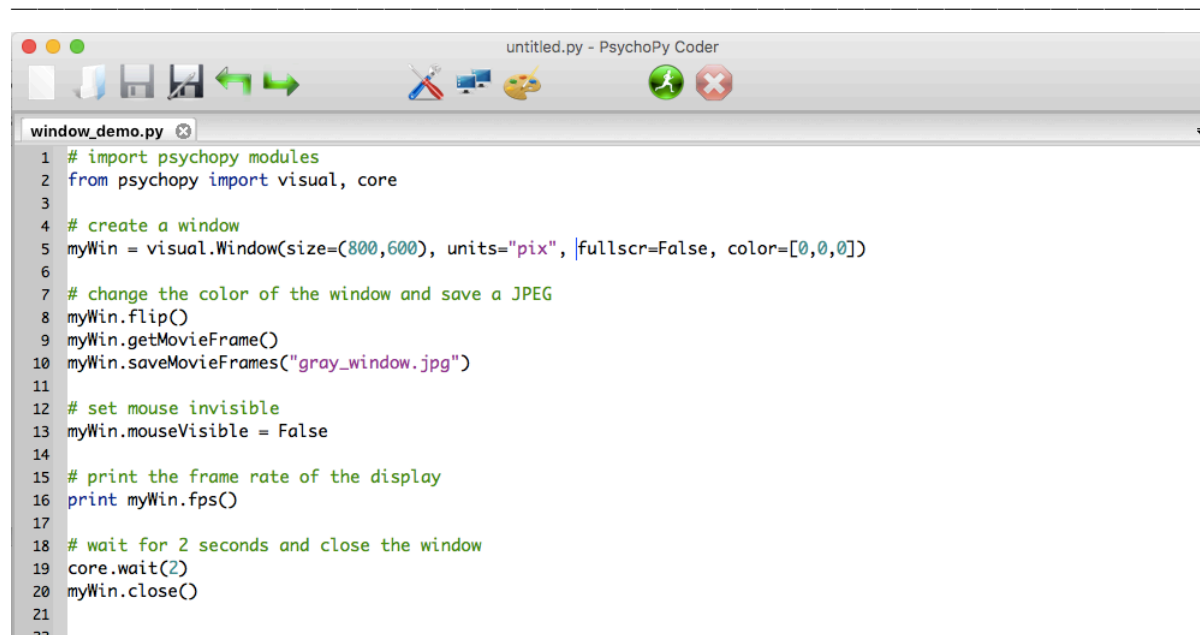


图 5.5 Window 模块实例。

5.6.2 多种视觉刺激

Psychopy 支持多种视觉刺激，包括简单的几何元素、图片、影片，以及多种视觉实验中经常使用的模式刺激（patterns；比如 checkerboard），下面简要介绍几种。

5.6.2.1 多边形（polygon）

绘制正多边形，这里的 edges 为边数。在下面的示例中，我们把 edges 设置为 360 来画一个近似的圆形。

```
visual.Polygon(win, edges=360, radius=0.5)
```

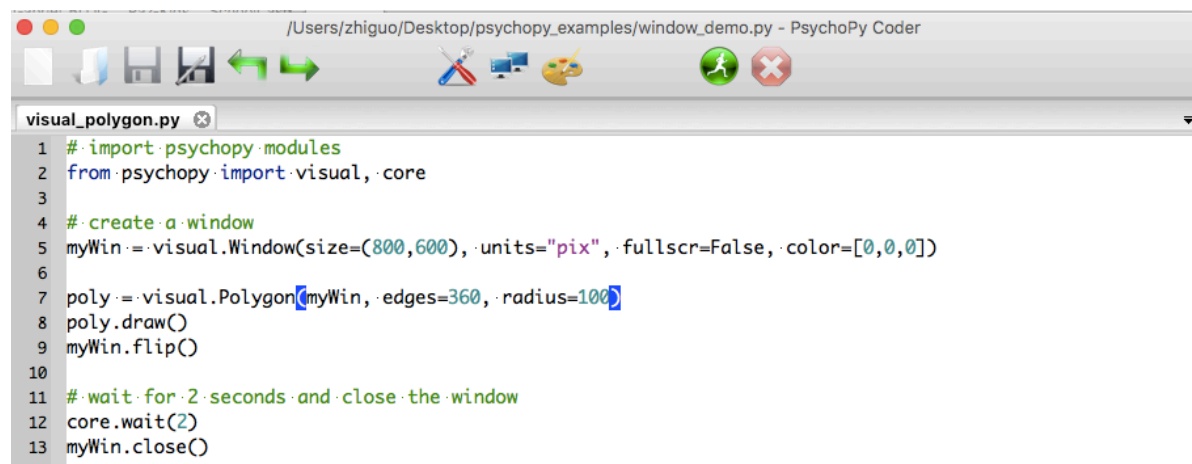


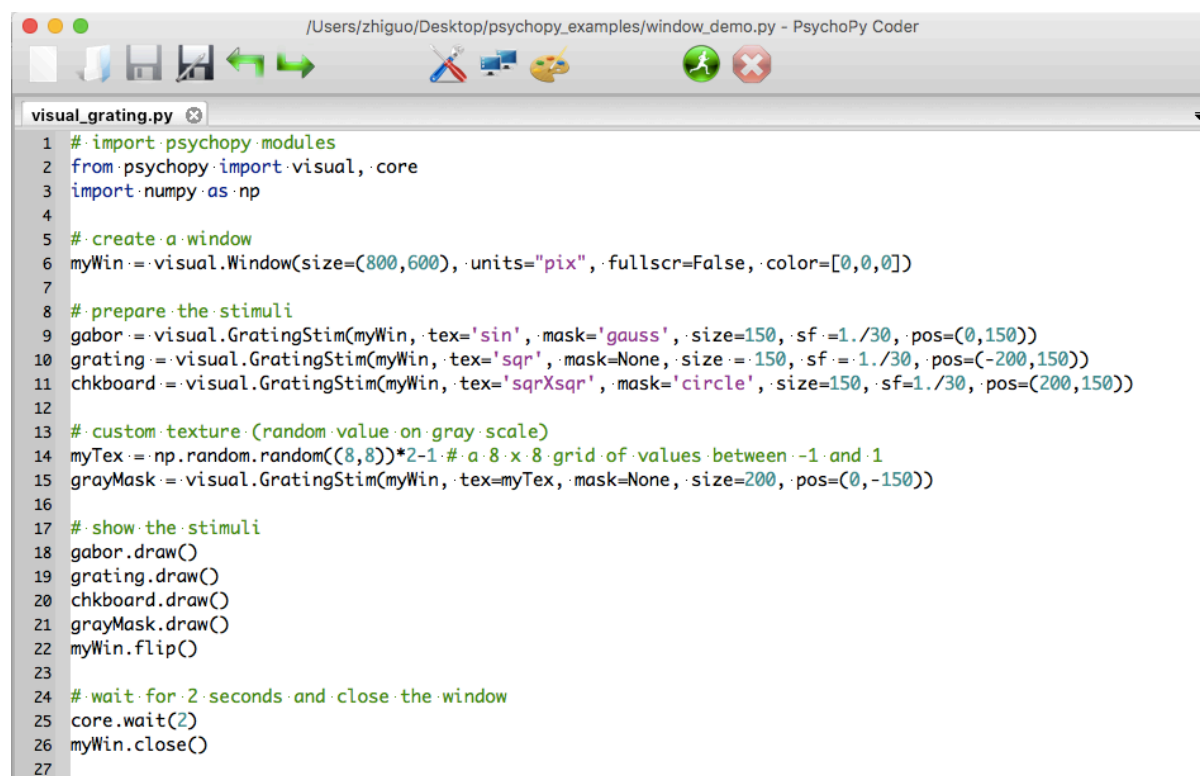
图 5.6 简单几何刺激实例。

5.6.2.2 光栅刺激 (GratingStim)

光栅刺激是视觉实验中经常使用的一类复杂刺激，比如 gabor、光栅，和棋盘刺激。需要注意的是，除了 Psychopy 自带的纹理（texture），我们还可以自定义纹理和 mask。Sebastiaan Mathot（OpenSesame 的作者）在他的 blog 里面给出了多个示例，读者可以在此链接获取更多的实例，<http://www.cogsci.nl/blog/tutorials/211-a-bit-about-patches-textures-and-masks-in-psychopy>。在下面的示例中，我们把 tex（纹理）设置为一个 8 x 8 的矩阵，矩阵的每个元素的取值在 -1 和 +1 之间。这个自定义的纹理会帮我们生成一个随机灰度的棋盘刺激。

```
visual.GratingStim(win, tex='sin', mask='none', units='', pos=(0.0, 0.0), size=None,
sf=None, ori=0.0, phase=(0.0, 0.0))
```

使用光栅刺激时需要注意屏幕“单位”对空间频率（spatial frequency）的取值的影响。在该实例中，屏幕的单位是“pix”（像素），我们设置 sf=1./20 因为我们不可能在单个像素内实现由黑到白的变换。但是如果屏幕的单位是“deg”（视角），我们就可以设置 sf=2.0，即在 1 度内由黑到白的变换发生两次（2 Hz）。



```
visual_grating.py
1 # import psychopy modules
2 from psychopy import visual, core
3 import numpy as np
4
5 # create a window
6 myWin = visual.Window(size=(800,600), units="pix", fullscr=False, color=[0,0,0])
7
8 # prepare the stimuli
9 gabor = visual.GratingStim(myWin, tex='sin', mask='gauss', size=150, sf=1./30, pos=(0,150))
10 grating = visual.GratingStim(myWin, tex='sqr', mask=None, size=150, sf=1./30, pos=(-200,150))
11 chkboard = visual.GratingStim(myWin, tex='sqrXsqr', mask='circle', size=150, sf=1./30, pos=(200,150))
12
13 # custom texture (random value on gray scale)
14 myTex = np.random.random((8,8))*2-1 # a 8 x 8 grid of values between -1 and 1
15 grayMask = visual.GratingStim(myWin, tex=myTex, mask=None, size=200, pos=(0,-150))
16
17 # show the stimuli
18 gabor.draw()
19 grating.draw()
20 chkboard.draw()
21 grayMask.draw()
22 myWin.flip()
23
24 # wait for 2 seconds and close the window
25 core.wait(2)
26 myWin.close()
27
```

图 5.7 光栅刺激实例。

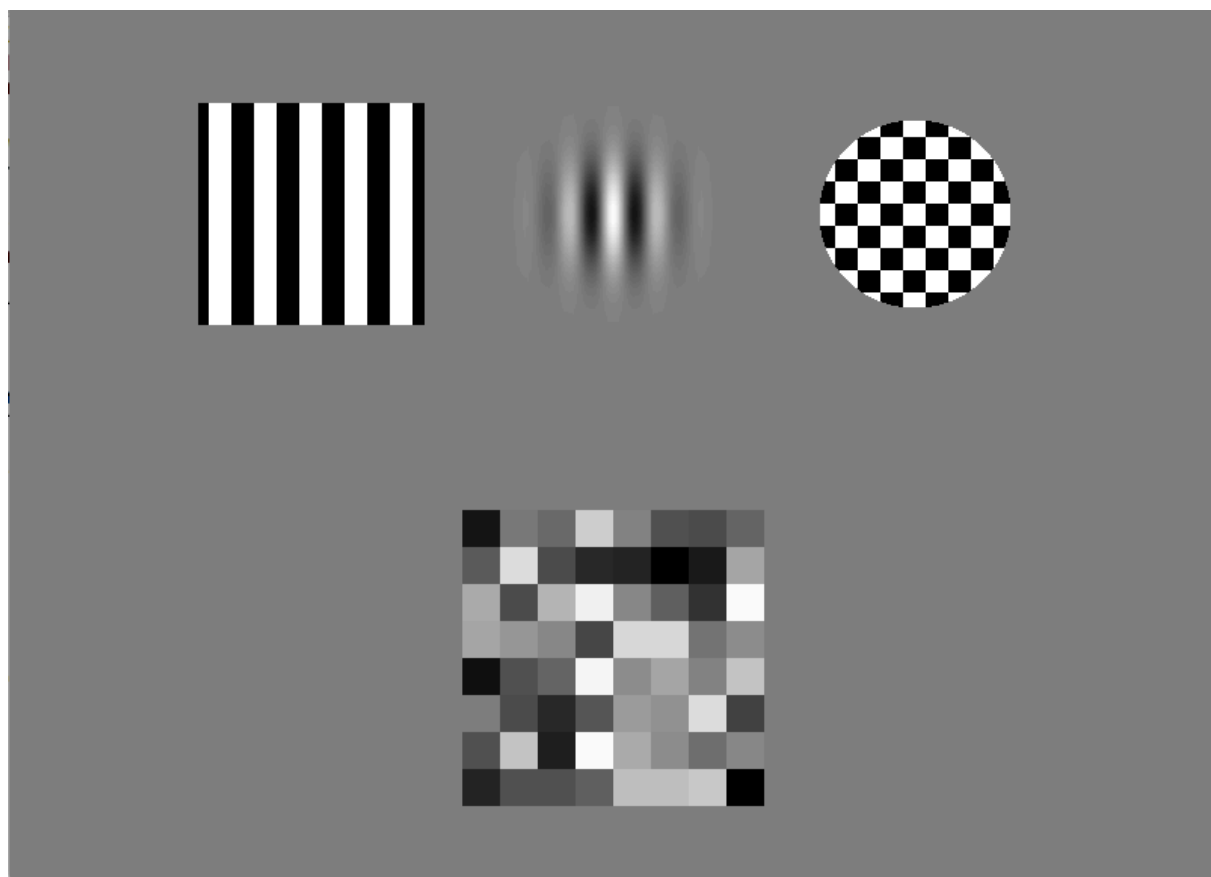


图 5.8 光栅刺激的运行结果。

5.6.2.3 放射状刺激 (RadialStim)

放射状刺激是以极坐标系为参照的一类刺激。在视觉影像研究中，我们常常需要使用旋转的楔形棋盘刺激（rotating wedge）或者/和收缩的环形棋盘刺激（contracting annulus）来估计视野在视觉皮层的映射。放射状刺激就是 Psychopy 为该目的而封装的一个功能模块。在下面的示例中，我们使用 Psychopy 的运算功能（operation）动态地改变楔形刺激的 ori 和 radialPhase 两个参数，从而让该刺激在旋转的同时相位也在动态变化。

```
visual.RadialStim(win, tex='sqrXsqr', mask='none', units='', pos=(0.0, 0.0), size=(1.0, 1.0),  
radialCycles=3, angularCycles=4, ori=0.0, visibleWedge=(0, 360))
```

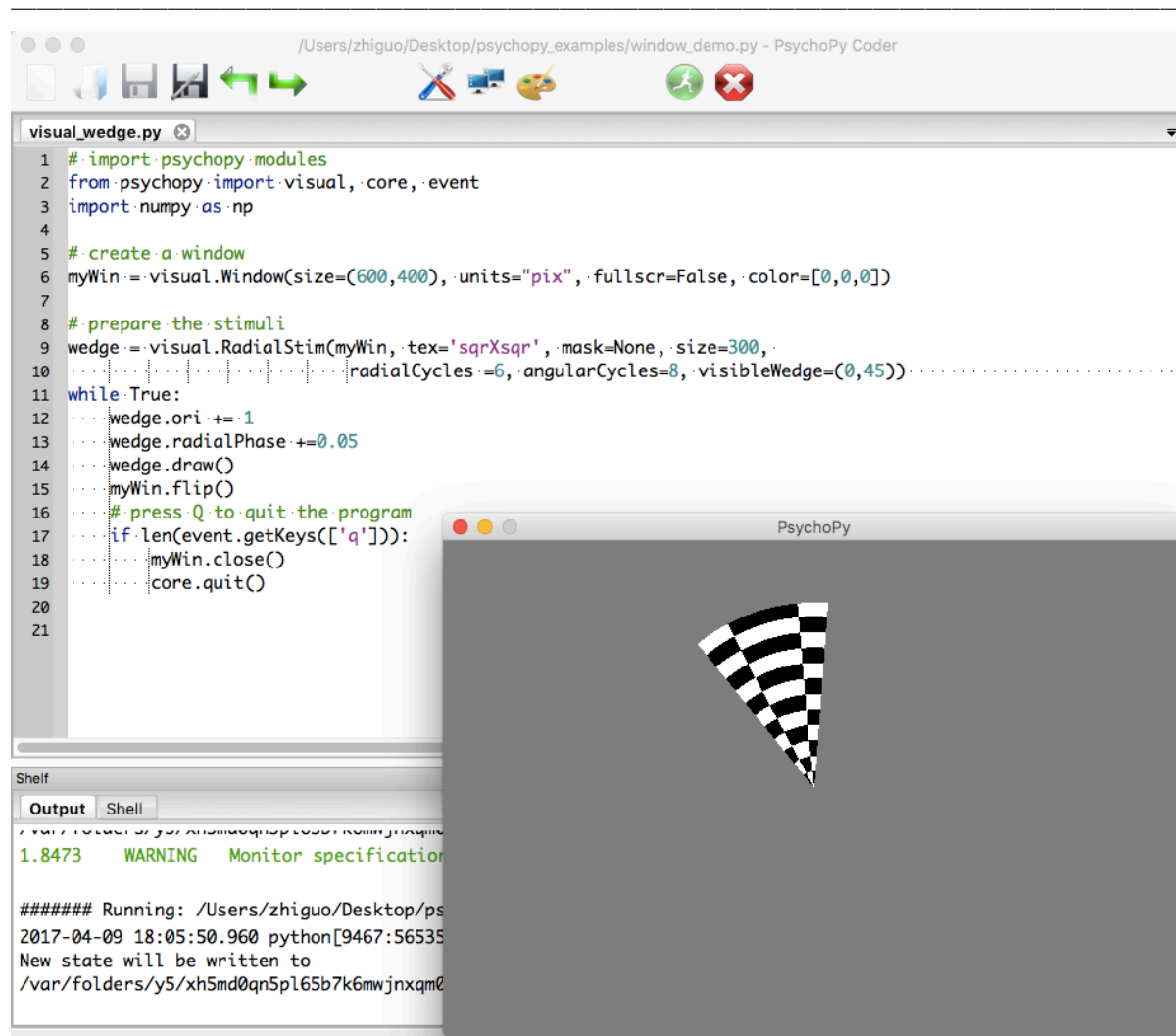


图 5.9 旋转的契形棋盘刺激。

5.6.3 光圈功能 (aperture)

这是 PsychoPy 的一个重要的功能，我们可以对屏幕上呈现的刺激应用一个光圈效果，仅让被试看到部分而不是全部刺激。需要注意的是，如果要使用该功能，我们必须首先需要激活 Window 的 allowStencil 参数。在下面的示例中，我们使用该功能呈现一个“移动窗口”，只有以鼠标位置为中心的矩形区域呈现给被试，其他屏幕区域则掩蔽起来。感兴趣的读者，可以把鼠标位置改为眼睛的注视位置（需连接眼动仪）即可实现阅读研究中常用的“移动窗口”任务。

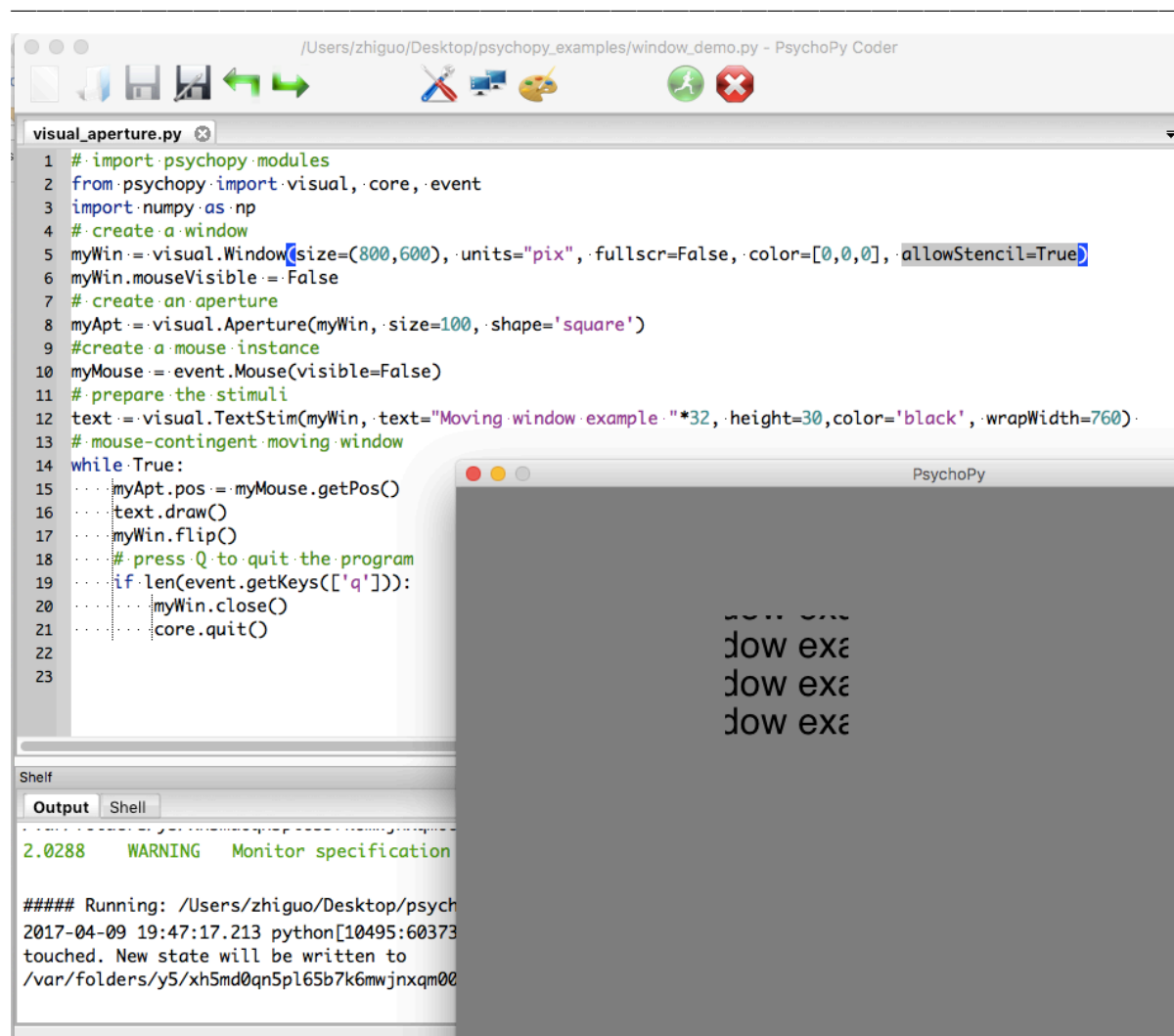


图 5.10 使用 Aperture 功能实现“移动窗口”。

5.6.4 多屏显示

Psychopy 的一个经常被用户遗忘的功能是多屏显示。该功能能给我们带来很多方便。比如，在磁共振实验中我可以用屏幕 A 呈现刺激给被试，屏幕 B 呈现实验进程信息给实验员。在比如，在多人交互的实验中，我们可使用多个屏幕和多个键盘实现多个被试间的“联网”对话交流。

```

win0 = visual.Window(size=(600,400), units="pix", screen=0)
win1 = visual.Window(size=(600,400), units="pix", screen=1)

```

在下面的示例中，我们在第一个屏幕（screen=0）上呈现一个旋转的契形刺激，然后在第二个屏幕（screen=1）上呈现一条消息，显示现在契形刺激旋转了多少圈。两个屏幕的刺激独立绘制，独立更新。

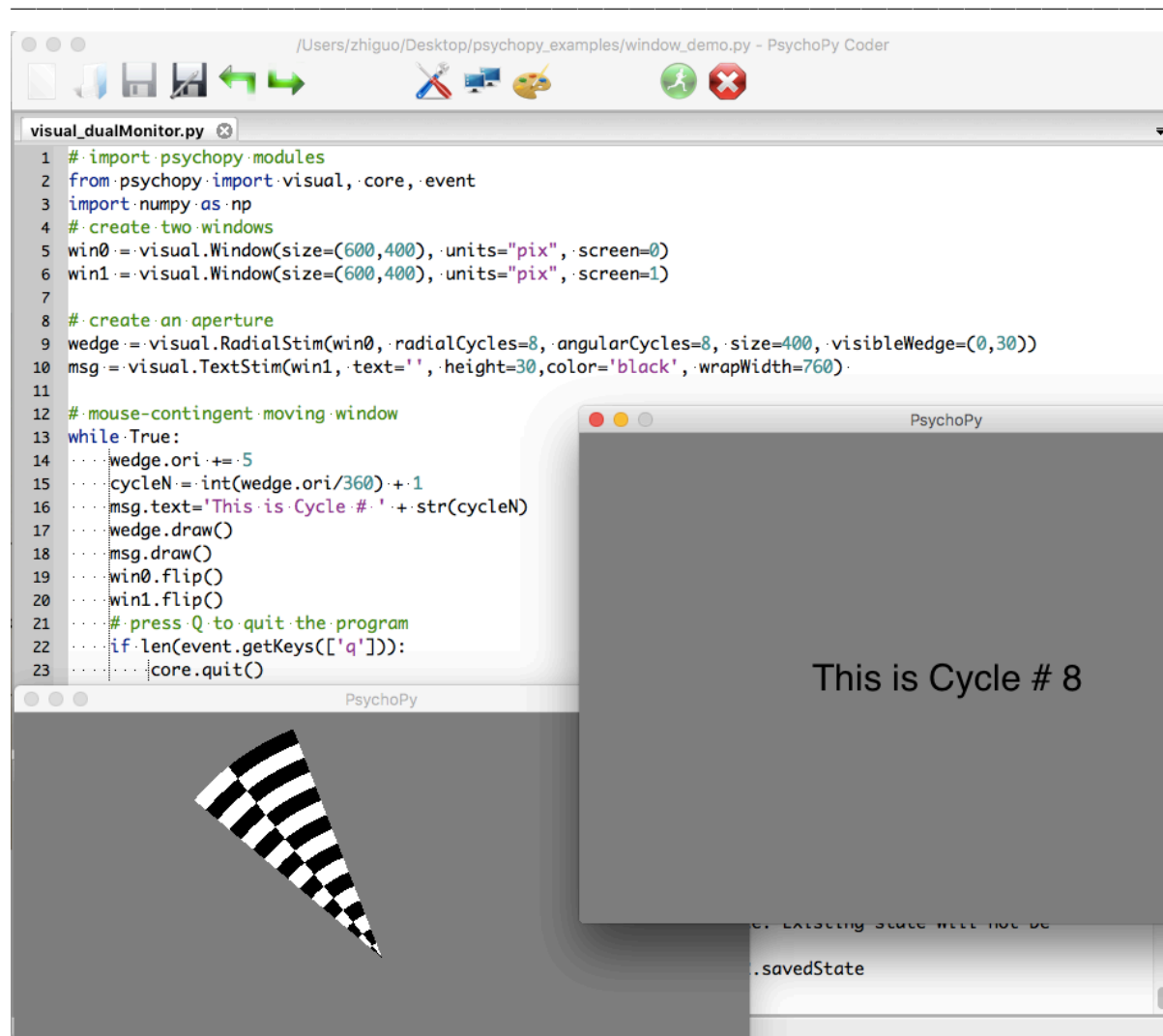


图 5.11 使用多个窗口。

5.7 试次 (trial) 控制

Psychopy 自带了一个控制试次的模块，但是作者认为有更直观的方法来控制试次的呈现和随机。在前面的 Pygame 章节，我们介绍了如何使用 List 来控制试次。鉴于部分用户可能会跳过 Pygame 相关的章节，这里我们再通过一个简单的 Psychopy 实验示例来简要介绍一下这个通用的方法。

基于我们的实验设计，通常可能的实验条件组合都有限。比如我们有两个变量 A 和 B，每个变量各有两个水平 A1、A2 和 B1、B2。那么我共有 $2 \times 2 = 4$ 中实验条件 (experimental cells)：[A1, B1]，[A1, B2]，[A2, B1]，[A2, B2]。我们把这些实验条件放入一个 list，`trial = [[A1, B1], [A1, B2], [A2, B1], [A2, B2]]`，就可以使用 list 的循环功能，逐个测试这些实验条件。如果每个实验条件需要重复 24 次，我们可以方便地生成一个

4 x 24 的试次列表，`trial_new = trial * 24`。如果需要对这个试次列表进行随机，我们可以调用 `random` 模块的 `shuffle` 函数，给试次列表中的元素随机排序，`random.shuffle(trial_new)`。然后我们就可以使用 `for` 循环来逐个测试试次列表中的所有试次。下面的简短示例代码就演示了这种简单的试次控制方法。

```
for t in trial_new:
    draw something, collect some response, save data to file
```

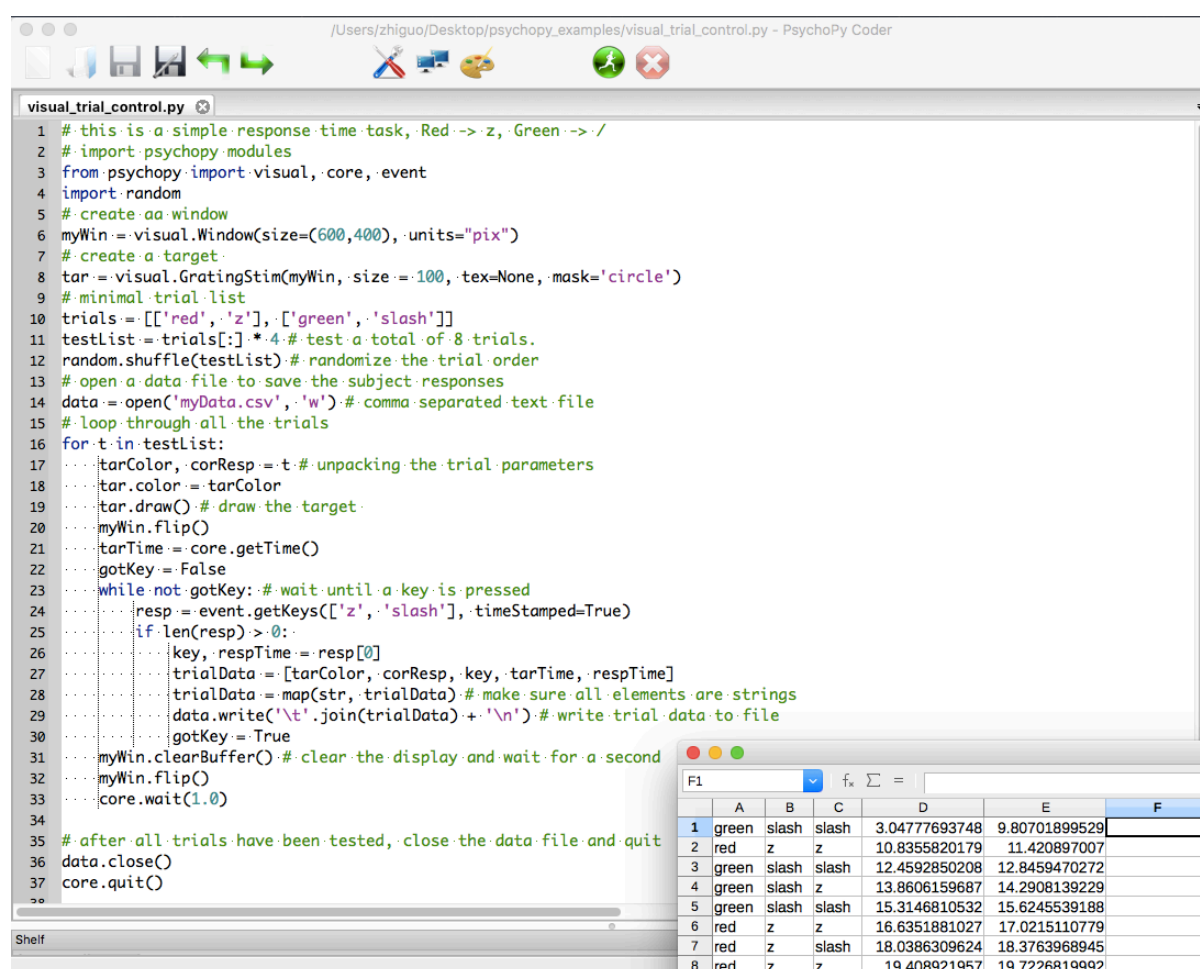


图 5.12 使用 `List` 控制试次。

这段简要的代码仅有 37 行（包括空行和注释），却实现了关键的实验功能，即刺激呈现，反应监测和数据记录。这充分体现了 Python 语法的简洁。