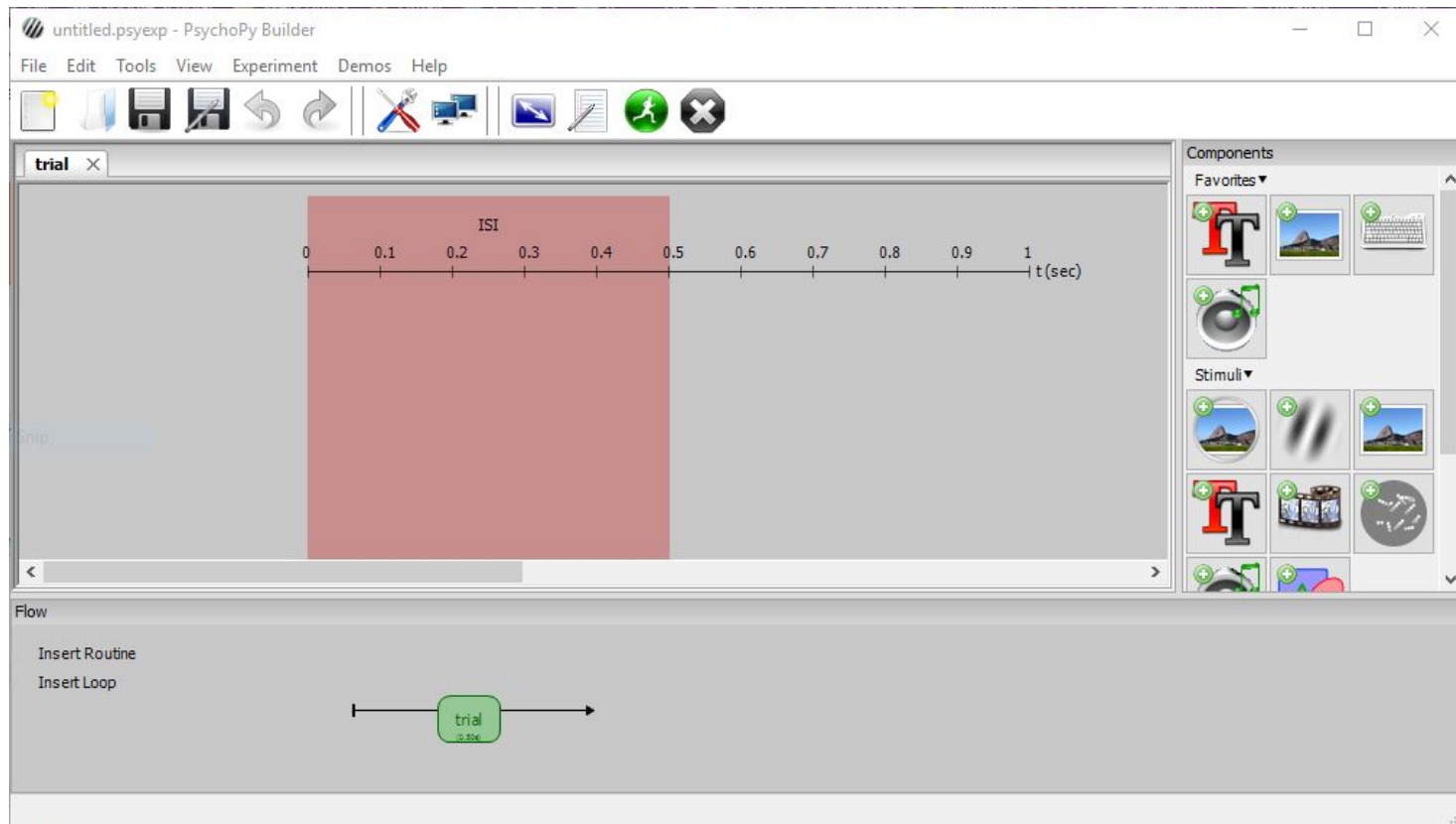

PsychoPy Tutorial

Why Use PsychoPy?

- Python is awesome
 - Simpler than MATLAB
 - Doesn't require a license
 - Those're the main pluses I think
-
- Can't be used for online applications, though
-

Builder



Routines and Loops

- In PsychoPy, a **routine** is like one block, or one type of screen.
 - An instruction screen is a routine. So is one trial.
 - Routines are full of COMPONENTS
 - A set of trials is a bunch of routines running through a **loop**.
 - A loop is a routine that changes based on given parameters.
-

Parameters

	A	B	C	D	E
1	image	prompt	correctAns		
2	imagesDK	blue	f		
3	imagesDK	red	f		
4	imagesDK	green	f		
5	imagesDK	purple	f		
6	imagesDK	blue	f		
7	imagesDK	red	f		
8	imagesDK	green	f		
9	imagesDK	purple	f		
10	imagesDK	blue	j		
11	imagesDK	red	j		
12	imagesDK	green	j		
13	imagesDK	purple	j		
14	imagesDK	blue	j		
15	imagesDK	red	j		
16	imagesDK	green	j		
17	imagesDK	purple	j		
18					
19					

Parameters

- Each row of your parameters file corresponds to one trial
 - In your routine, you call these variables like R factors:
\$columnHeader
-

Loop Properties

Loop Properties ✕

Name

loopType ▾

Is trials ☒

random seed \$

nReps \$

Selected rows \$

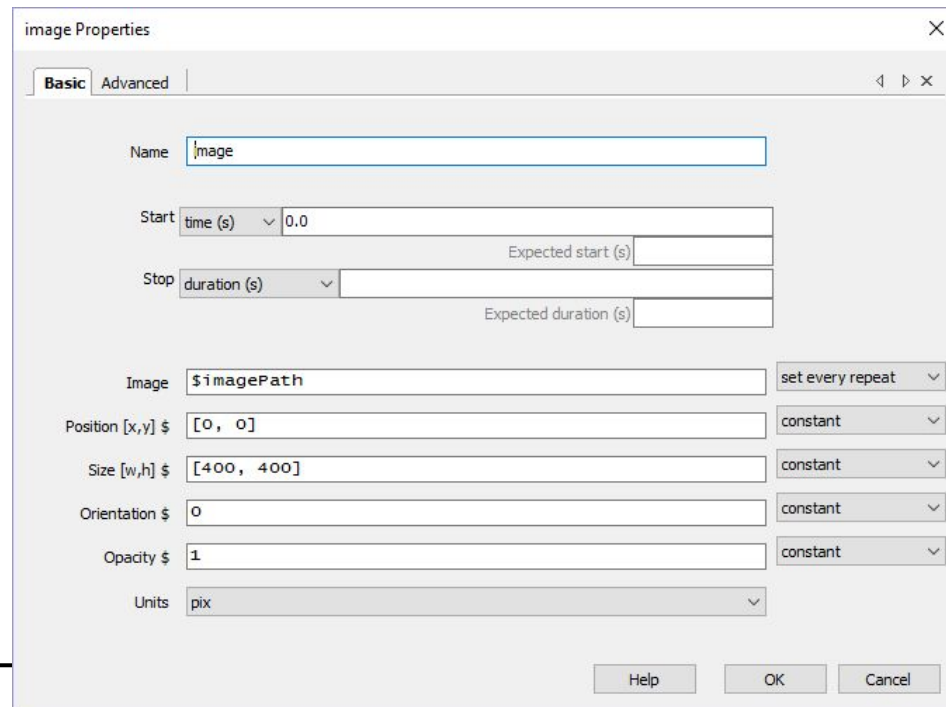
Conditions Browse...

16 conditions, with 3 parameters
[correctAns, image, prompt]

Help OK Cancel

Stimuli

When making your stimuli, you reference the conditions.



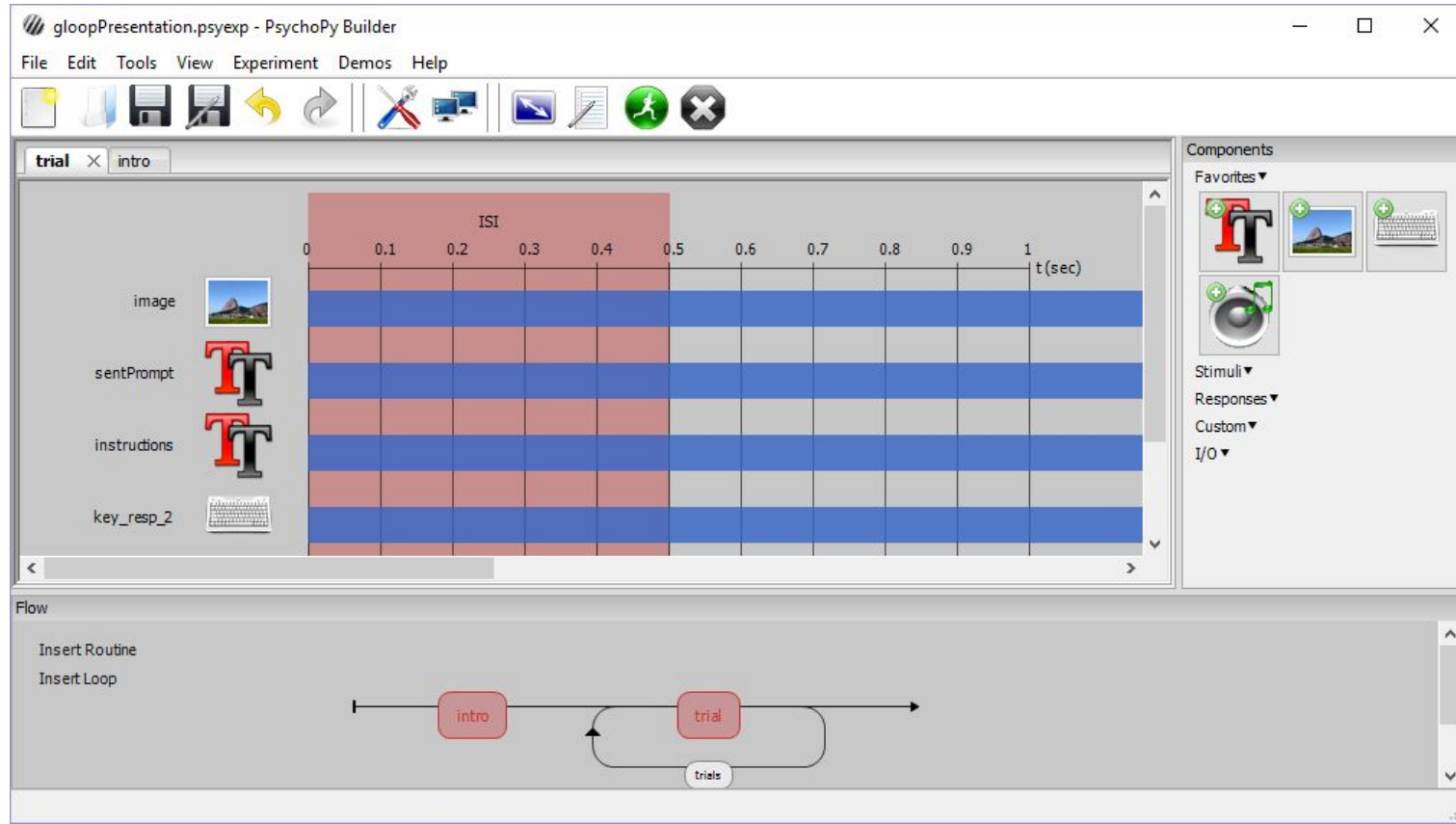
The screenshot shows a software window titled "image Properties" with a close button (X) in the top right corner. Below the title bar are two tabs: "Basic" (selected) and "Advanced". To the right of the tabs are navigation icons: a left arrow, a right arrow, and a close button (X).

The "Basic" tab contains the following settings:

- Name:** A text field containing the text "image".
- Start:** A dropdown menu set to "time (s)" with a value of "0.0". To its right is a text field labeled "Expected start (s)".
- Stop:** A dropdown menu set to "duration (s)" with an empty text field next to it. To its right is a text field labeled "Expected duration (s)".
- Image:** A text field containing "\$imagePath". To its right is a dropdown menu set to "set every repeat".
- Position [x,y]:** A text field containing "[0, 0]". To its right is a dropdown menu set to "constant".
- Size [w,h]:** A text field containing "[400, 400]". To its right is a dropdown menu set to "constant".
- Orientation:** A text field containing "0". To its right is a dropdown menu set to "constant".
- Opacity:** A text field containing "1". To its right is a dropdown menu set to "constant".
- Units:** A dropdown menu set to "pix".

At the bottom of the window are three buttons: "Help", "OK", and "Cancel".

And after some work...



Demonstration

Why code it yourself?

So, say you have a problem with your builder code, or need to do something the builder can't.

So you go look at your builder code and...



gloopPresentation.py ×

```
37 ...originPath=None,
38 ...savePickle=True, .savewideText=True,
39 ...dataFileName=filename)
40 logging.console.setLevel(logging.WARNING) .#.this outputs to the screen, not a file
41
42 endExpNow = False .#.flag for 'escape' or other condition => quit the exp
43
44 #Start Code-- component code to be run before the window creation
45
46 #Setup the window
47 win = visual.Window(size=(1366, 768), fullscr=True, screen=0, allowGUI=False, allowStencil=False,
48 ...monitor='testMonitor', color=[0.004,0.004,0.004], colorSpace='rgb',
49 ...blendMode='avg', useFBO=True,
50 ...)
51 #store frame rate of monitor if we can measure it successfully
52 expInfo['frameRate']=win.getActualFrameRate()
53 if expInfo['frameRate']!=None:
54 ...frameDur = 1.0/round(expInfo['frameRate'])
55 else:
56 ...frameDur = 1.0/60.0 #couldn't get a reliable measure so guess
57
58 #Initialize components for Routine "intro"
59 introClock = core.Clock()
60 text = visual.TextStim(win=win, ori=0, name='text',
61 ...text=u'welcome to the experiment!\n\nToday you will press buttons.\n\nPress any button to start.', ...font=u'Arial',
62 ...pos=[0,0], height=0.15, wrapwidth=None,
63 ...color=u'Black', colorSpace='rgb', opacity=1,
64 ...depth=0.0)
65
66 #Initialize components for Routine "trial"
67 trialClock = core.Clock()
```

Shelf

Output Shell

welcome to PsychoPy2!

v1.83.04

Woah

It's a lot.

Not incomprehensible, which is impressive, but certainly not as customizable as self-coded.

Coding

- There's a coder built into the Psychopy program
 - But I like Sublime Text. It looks nicer.
 - Atom is another great alternative
-

```
1 from __future__ import division
2 import time
3 import sys
4 import random
5 import csv
6 import math
7 from numpy import linspace
8 from psychopy import visual,event,core
9
10 win = visual.Window([400,400], units='pix', monitor='testMonitor', color='black')
11
12 square = visual.Rect(win,linewidth=0,fillColor="blue",size=[.2,.2],pos=[0,-.35], units="height")
13 square2 = visual.Rect(win,linewidth=0,fillColor="blue",size=[.2,.2],pos=[0,.15], units="height")
14 grass = visual.Rect(win,linewidth=0,fillColor="green",size=[2,.3],pos=[0,-.45], units="height")
15
16 numframes = 200
17 frames = linspace(0,math.pi,numframes)
18 count=0
19 minimum = -.35
20 maximum = .15
21 height = maximum + -minimum
22 while not event.getKeys(keyList=['q']):
23     if count>numframes:
24         count=0
25
26     x=frames[count]
27     spos = height * math.sin(x)
28     spos += minimum
29     #ypos = starHeight * sin(x);
30     square.setPos([0,spos])
31     grass.draw()
32     square.draw()
33     win.flip()
34     count += 1
35
36 sys.exit()
```

Packages

There are a lot of **packages** of code in Python, which can give you a lot of useful functions.

Use **import** to bring in the functions from those packages. This is the first thing you should do.

```
from __future__ import division
import time
import sys
import random
import csv
import math
from numpy import linspace
from psychopy import visual, event, core
```

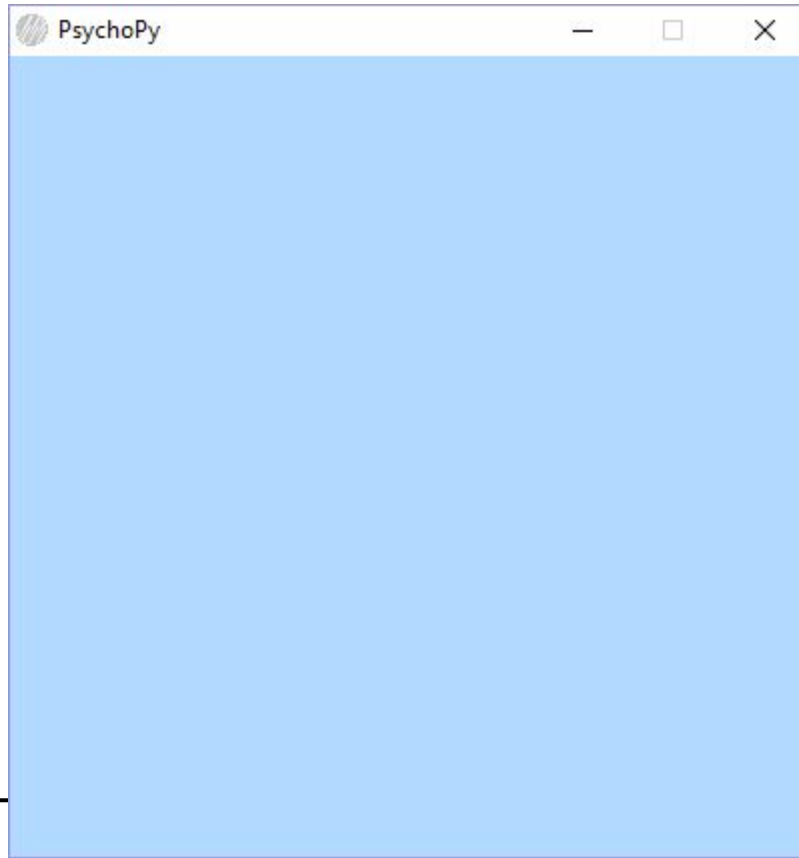
Opening a window

Like in Matlab, you start by creating the **window** the experiment will run in.

You can set a bunch of **parameters** like size, the units of the size, and the background color.

```
1  from __future__ import division
2  from psychopy import visual, event, core
3  import time
4  import sys
5  import random
6  import csv
7  import math
8  from numpy import linspace
9
10 win = visual.Window([400,400], units='pix', monitor='testMonitor', color=[.4, .7, 1],
11                    colorSpace="rgb"])
12
13 while not event.getKeys(keyList=['q']):
14     continue
15
16 sys.exit()
```

Wow! Very blue. Progress.



Drawing

Also similar to Matlab, you can create shapes, and then draw them to the screen.

NOTE: Each time you use `win.flip()`, it **erases** everything drawn before the previous flip, and **draws** everything drawn after the previous flip.

Naming

Fun fact for programming in any language: **variables names are important!** They should be long enough to say what they are, but not inconveniently long.

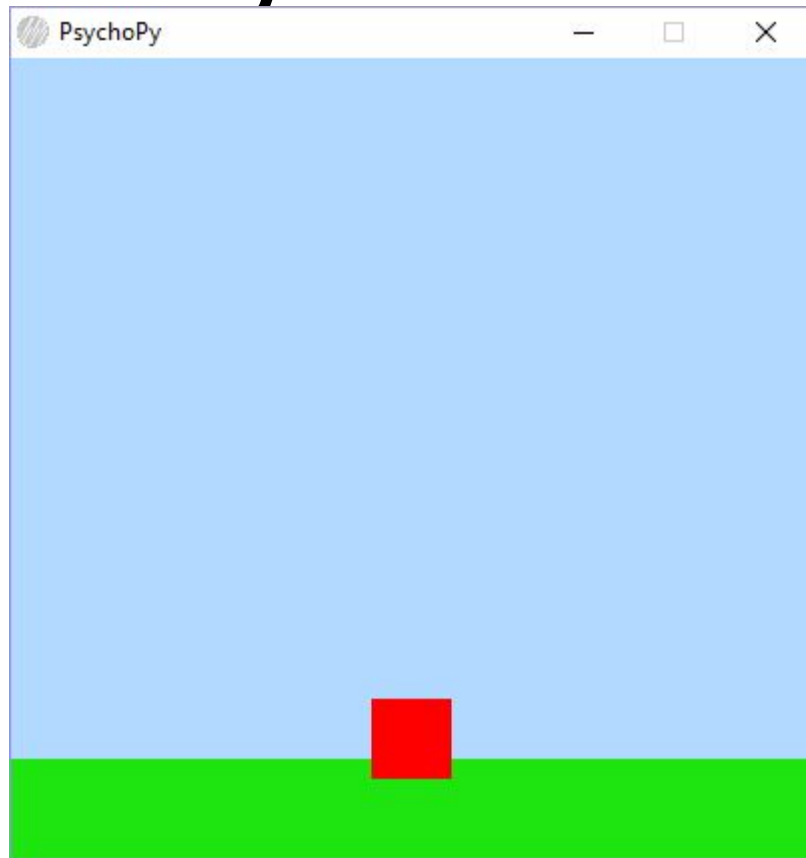
And a lot of programming apps like Sublime Text will help you fill things in as you type.

Pop quiz: Can you guess what I'm drawing in the code on the next slide?

What am I drawing?

```
9
10 win = visual.Window([400,400], units='pix', monitor='testMonitor', color=[.4, .7, 1],
11     colorSpace="rgb")
12
13 square = visual.Rect(win, lineWidth=0, fillColor="red", size=[.2,.2], pos=[0, -.35],
14     units="height")
15 grass = visual.Rect(win, lineWidth=0, fillColor=[30, 229, 15], fillColorSpace="rgb255",
16     size=[2,.3], pos=[0, -.45], units="height")
17
18 while not event.getKeys(keyList=['q']):
19     grass.draw()
20     square.draw()
21     win.flip()
22
23 sys.exit()
```

Hooray!



Comments

The next steps are a bit more complicated than static shapes, so I've **commented** my code to make it easier to understand!

Comments are used to describe parts of your code that might be hard to understand. (I've commented more than usual, since it's some mathy stuff that might be hard to track.)

Your comments, unlike the rest of your code, are **not run**! So you can also comment out lines of code that you don't need anymore, but want to remember.

Comments

```
18 numframes = 200
19 #numframes is the number of frames for each jump
20 framesArray = linspace(0,math.pi,numframes)
21 #frames is the list of points used to calculate the square's height.
22 #It's an array that contains numframes points, spaced equally from 0 to pi
23 #This is to accomodate the sine curve used to plot these points.
24 count=0
25 #count is an iterater
26 minimum = -.35
27 #The minimum point is the lowest point the square will be at.
28 maximum = .15
29 #The maximum point is the highest point the square will be at.
30 height = abs(maximum) + abs(minimum)
31 #the height is the full distance the square will traverse
32 |
```

And finally, the actual movement

```
32
33 while not event.getKeys(keyList=['q']):
34     #We'll stay in this while loop until the user presses 'q'
35     if count>=numframes:
36         count=0
37         #Resets the iterator
38         core.wait(.25)
39     x=framesArray[count]
40     #x grabs the corresponding point between 0 and pi
41     squarepos = (height * math.sin(x)) + minimum
42     #The position for the square is:
43     #   the point on the sine curve (which is between 0 and 1)
44     #   *the maximum height (so how high it is depends on the sine curve fraction)
45     #   +minimum (so that it accounts for the position is started at; in our case, it's lower)
46     square.setPos([0,squarepos])
47     #We change the position of the square object to new position
48     grass.draw()
49     square.draw()
50     #Then redraw our shapes
51     win.flip()
52     #And then put them on-screen!
53     count += 1
54     #Finally, we increment our iterator so we calculate a new number on the next run
55
56 sys.exit()
```

Demonstration!

(note to self: show the experiment)

Part 2: Responses and Parameters

Interfacing with Data

	A	B	C
1	height	color	
2	0.5	blue	
3	0.3	red	
4	0.7	purple	
5	0.4	red	
6	0.6	purple	
7	0.5	blue	
8			

- It mostly uses Python's csv package
 - You can Google it to find out everything it can do
 - I'll be using the csv dictionary writer, and the csv writer.
 - When reading the input, each row will be a **dictionary**, which maps keys to values
 - In our case, the column header will be the **key** and the parameters for that trial will be the **value**
-

Csv Readers and Writers

Step 1: Open a file

Step 2: Create a reader or writer for that file

```
32  
33 parametersfile = open('eventsparameters.csv', 'r')  
34 parametersreader = csv.DictReader(parametersfile)  
35 responsefile = open('eventsresponses.csv', 'wb')  
36 responsefields = ['Response', 'RT']  
37 responsewriter = csv.DictWriter(responsefile, responsefields)  
38 responsewriter.writeheader()  
39 timer = core.Clock()  
40
```

Also, at the end, close the file! filename.close() will do it!

Response Screen

```
69
70     #Response screen
71     background = visual.Rect(win,linewidth=0,fillColor="black",size=[400,400],pos=[0,0],
72                               units="height")
73     #Cover up the background with black; the square should be the same size as the window
74     responseText = visual.TextStim(win,text='Was that cool?', height=40, color='white',pos=[0,0])
75     promptText = visual.TextStim(win,text='Press f for yes and j for no',
76                                   height=20, color='white',pos=[0,-40])
77     background.draw()
78     responseText.draw()
79     promptText.draw()
80     #Create the text, then draw it.
81     win.flip()
82     timer.reset()
83     #start the timer
84     while True:
85         response=event.waitKeys(keyList=['f','j'])[0]
86         break
87         #Wait for a response, and leave the loop when you have it
88     rt = timer.getTime()
89     responsewriter.writerow({'Response': response, 'RT': rt})
90     #Write the row of data
91
```

Close up shop!

```
94  
95 parametersfile.close()  
96 responsefile.close()  
97 sys.exit()
```

Final Result!

(Rebecca show the thing)

Citing

Apparently you're supposed to cite their paper (Peirce 2007; 2009) if you use PsychoPy in published work
