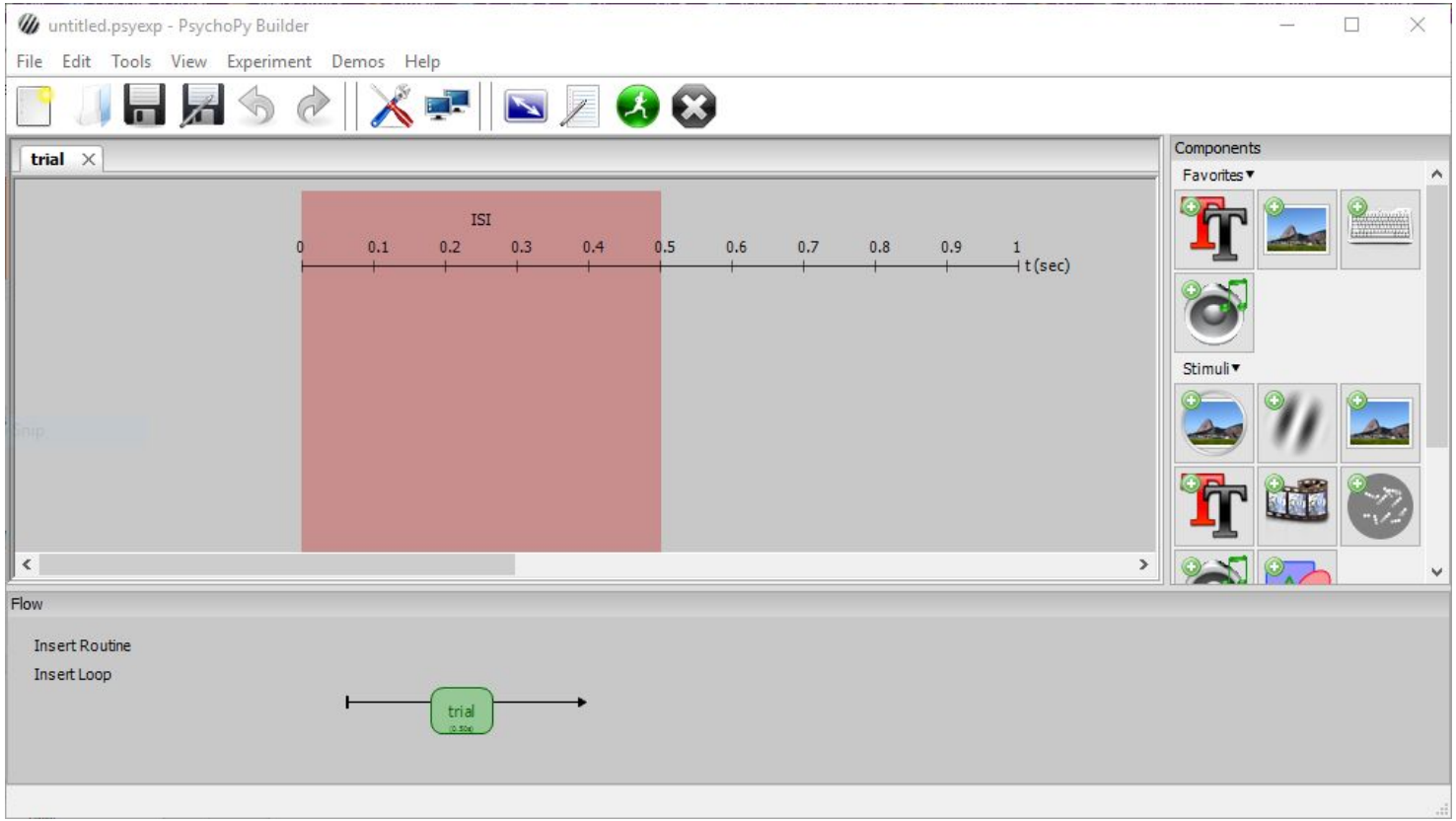
PsychoPy Tutorial

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

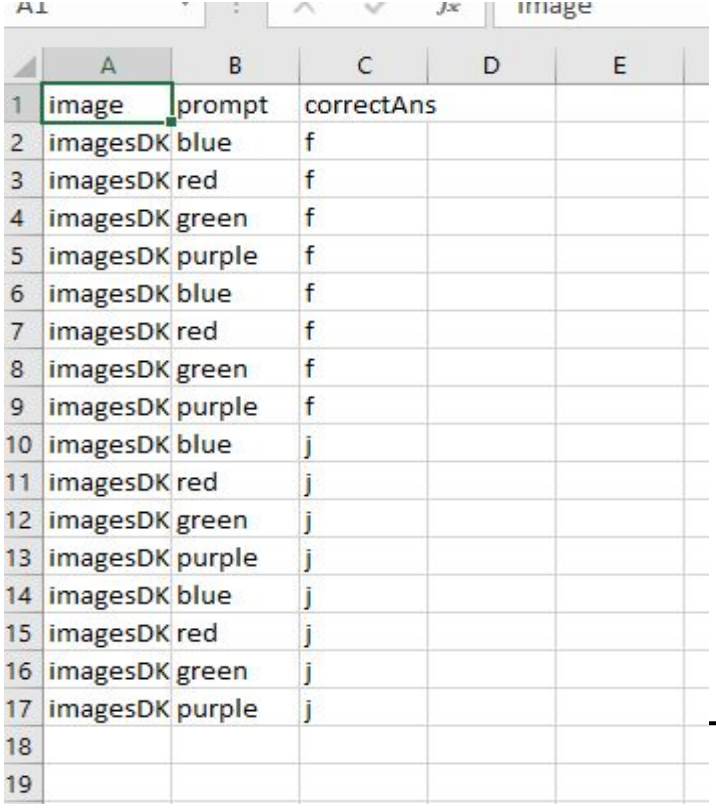
Builder


untitled.psyexp - PsychoPy Builder
File Edit Tools View Experiment Demos Help
trial
ISI
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
t (sec)
Components
Favorites
Stimuli
Flow
Insert Routine
Insert Loop
trial

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

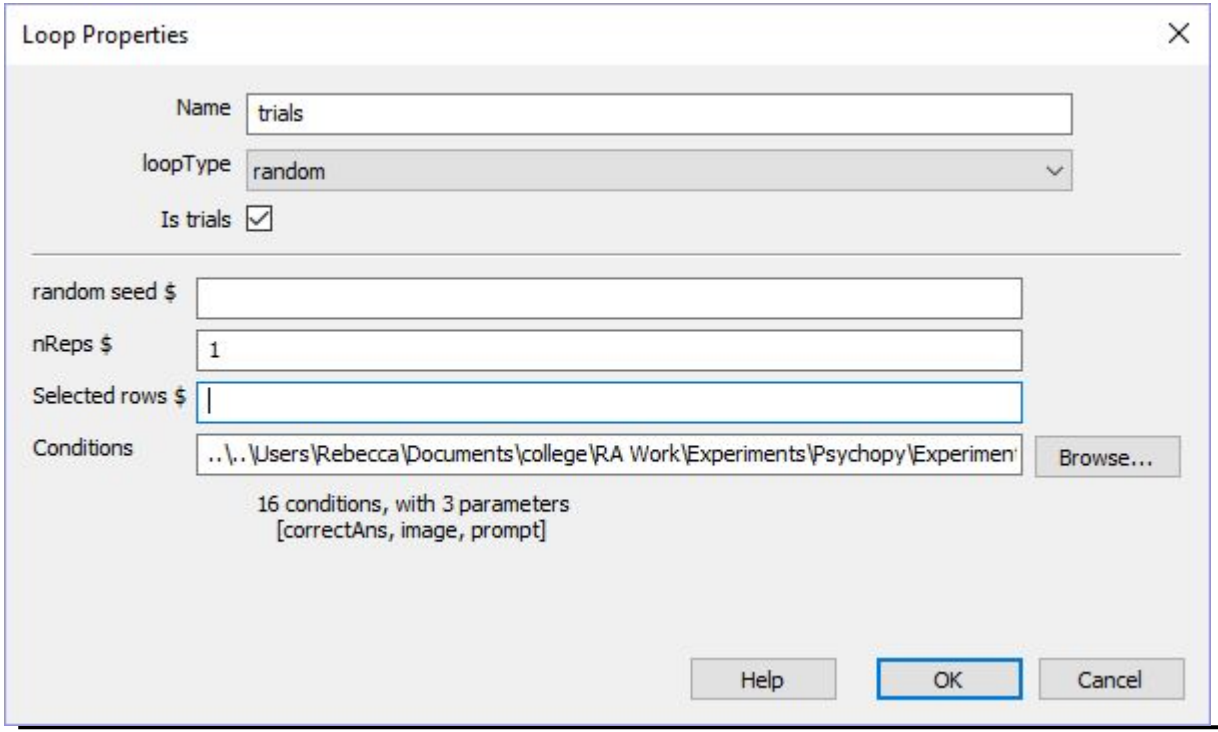


	A	B	C	D	E
1	image	prompt	correctAns		
2	imagesDK	blue	f		
3	imagesDK	red	f		
4	imagesDK	green	f		
5	imagesDK	purple	f		
6	imagesDK	blue	f		
7	imagesDK	red	f		
8	imagesDK	green	f		
9	imagesDK	purple	f		
10	imagesDK	blue	j		
11	imagesDK	red	j		
12	imagesDK	green	j		
13	imagesDK	purple	j		
14	imagesDK	blue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

Parameters

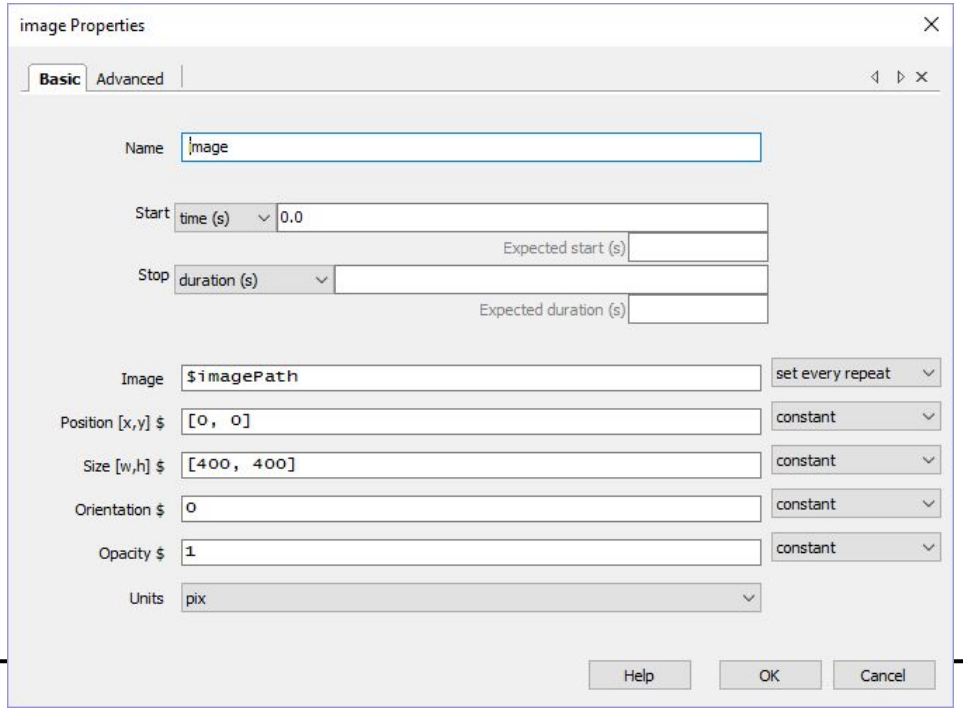
- Each row of your parameters file corresponds to one trial
- In your routine, you call these variables like R factors: $columnHeader

Loop Properties



Stimuli

When making your stimuli, you reference the conditions.



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...

C:\Users\Rebecca\Documents\college\RA Work\Experiments\Psychopy\Experiments\gloopPresentation.py - PsychoPy Coder

File Edit Tools View Demos Help

gloopPresentation.py

```
    originPath=None,
    savePickle=True, saveWideText=True,
    dataFileName=filename)
logging.console.setLevel(logging.WARNING)  # this outputs to the screen, not a file

endExpNow = False  # flag for 'escape' or other condition => quit the exp

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

# Setup the Window
win = visual.Window(size=(1366, 768), fullscr=True, screen=0, allowGUI=False, allowStencil=False,
    monitor='testMonitor', color=[0.004,0.004,0.004], colorSpace='rgb',
    blendMode='avg', useFBO=True,
    )
# store frame rate of monitor if we can measure it successfully
expInfo['frameRate']=win.getActualFrameRate()
if expInfo['frameRate']!=None:
    frameDur = 1.0/round(expInfo['frameRate'])
else:
    frameDur = 1.0/60.0 # couldn't get a reliable measure so guess

# Initialize components for Routine "intro"
introClock = core.Clock()
text = visual.TextStim(win=win, ori=0, name='text',
    text=u'Welcome to the experiment!\n\nToday you will press buttons.\n\nPress any button to start.',    font=u'Arial',
    pos=[0, 0], height=0.15, wrapWidth=None,
    color=u'Black', colorSpace='rgb', opacity=1,
    depth=0.0)

# Initialize components for Routine "trial"
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

C:\Users\Rebecca\Documents\college\RA Work\Experiments\Psychopy\eventsPresentation.py - Sublime Text 2 (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

disp_square.py | Exercise2_namesRSVP.py | eventsPresentation.py | events1v2.m | run.py | jspsych.js

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

win = visual.Window([400,400], units='pix', monitor='testMonitor', color='black')

square = visual.Rect(win,lineWidth=0,fillColor="blue",size=[.2,.2],pos=[0,-.35], units="height")
square2 = visual.Rect(win,lineWidth=0,fillColor="blue",size=[.2,.2],pos=[0,.15], units="height")
grass = visual.Rect(win,lineWidth=0,fillColor="green",size=[2,.3],pos=[0,-.45], units="height")

numframes = 200
frames = linspace(0,math.pi,numframes)
count=0
minimum = -.35
maximum = .15
height = maximum + -minimum
while not event.getKeys(keyList=['q']):
    if count>=numframes:
        count=0

    x=frames[count]
    spos = height * math.sin(x)
    spos += minimum
    #ypos = starHeight * sin(x);
    square.setPos([0,spos])
    grass.draw()
    square.draw()
    win.flip()
    count += 1

sys.exit()
```

Line 20, Column 14 Tab Size: 4 Python

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
```

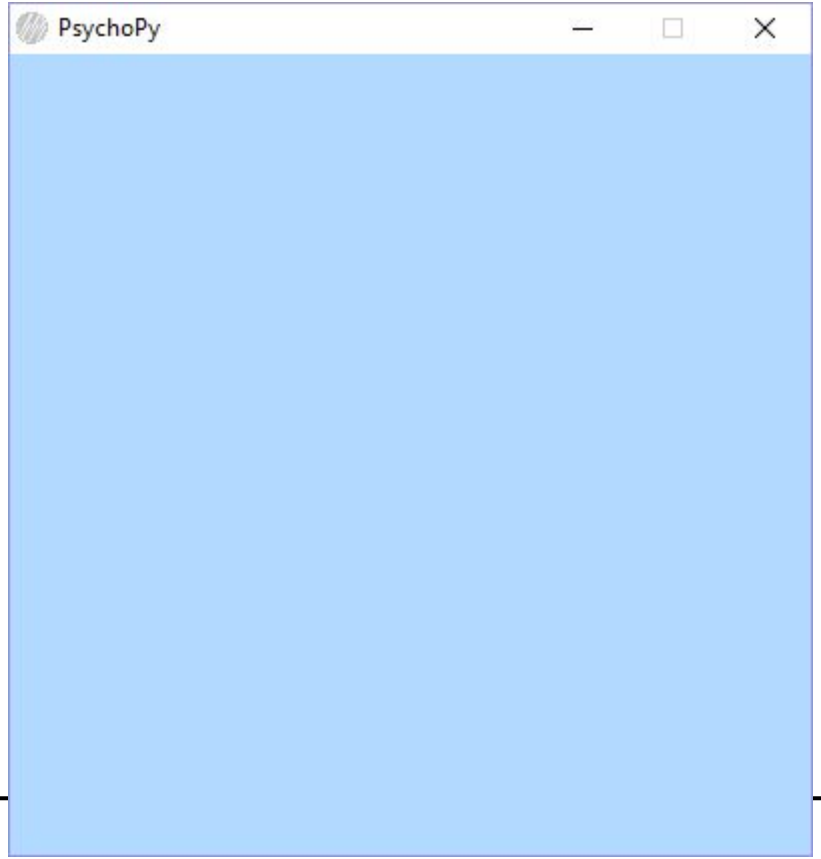
Opening a window

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: **variable names are important!** They should be long enough to say what they are, but not inconveniently long.

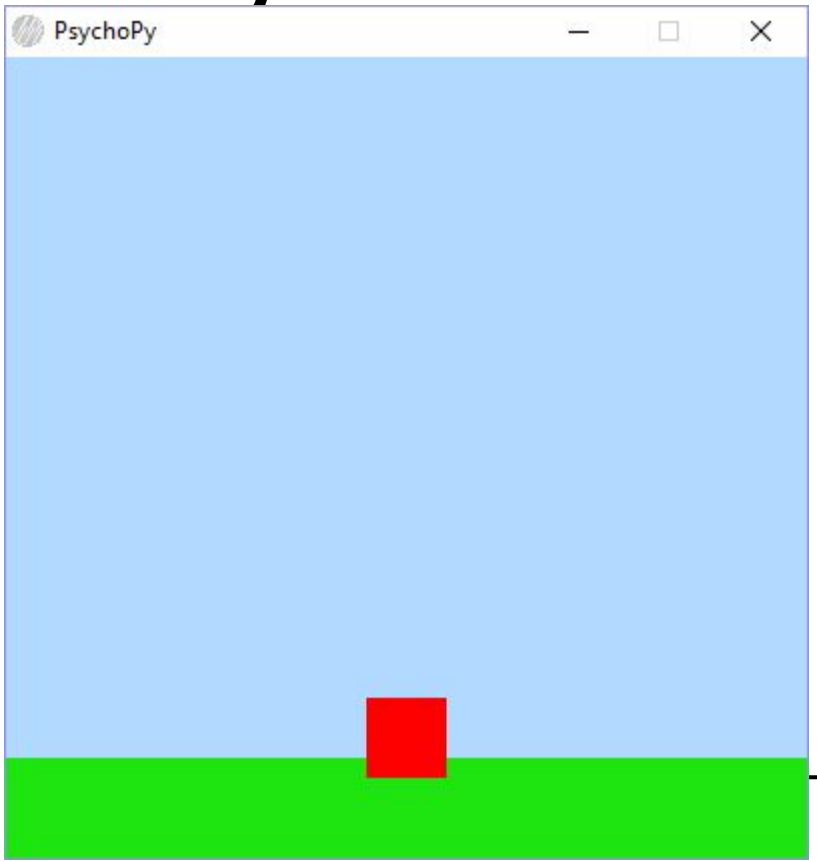
And a lot of programming apps like Sublime Text will help you fill things in as you type.

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

What am I drawing?

```
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

```
numframes = 200
#numframes is the number of frames for each jump
framesArray = linspace(0,math.pi,numframes)
#frames is the list of points used to calculate the square's height.
#It's an array that contains numframes points, spaced equally from 0 to pi
#This is to accomodate the sine curve used to plot these points.
count=0
#count is an iterater
minimum = -.35
#The minimum point is the lowest point the square will be at.
maximum = .15
#The maximum point is the highest point the square will be at.
height = maximum - minimum
#the height is the full distance the square will traverse
```

And finally, the actual movement

```
while not event.getKeys(keyList=['q']):
#We'll stay in this while loop until the user presses 'q'
    if count>=numframes:
        count=0
        #Resets the iterater
        core.wait(.25)
    x=framesArray[count]
    #x grabs the corresponding point between 0 and pi
    squarepos = (height * math.sin(x)) + minimum
    #The position for the square is:
    #   the point on the sine curve (which is between 0 and 1)
    #  *the maximum height (so how hight it is depends on the sine curve fraction)
    #  +minimum (so that it accounts for the position is started at; in our case, it's lower)
    square.setPos([0,squarepos])
    #We change the position of the square object to new position
    grass.draw()
    square.draw()
    #Then redraw our shapes
    win.flip()
    #And then put them on-screen!
    count += 1
    #Finally, we increment our iterator so we calculate a new number on the next run
```

Demonstration!

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 dictionary reader.
- 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

```
parametersfile = open('eventsparameters.csv', 'r')
parametersreader = csv.DictReader(parametersfile)
responsefile = open('eventsresponses.csv', 'wb')
responsefields = ['Response', 'RT']
responsewriter = csv.DictWriter(responsefile, responsefields)
responsewriter.writeheader()
timer = core.Clock()
```

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

For loops

```
41  for trial in parametersreader:
42      height = float(trial['height'])
43      square.fillColor = trial['color']
44      #Set parameters
45      count = 0
46      #Reset the iterater at the start of each trial
```

Response Screen

```
    #Response screen
    background = visual.Rect(win,lineWidth=0,fillColor="black",size=[400,400],pos=[0,0],
    units="height")
    #Cover up the background with black; the square should be the same size as the window
    responseText = visual.TextStim(win,text='Was that cool?', height=40, color='white',pos=[0,0])
    promptText = visual.TextStim(win,text='Press f for yes and j for no',
        height=20, color='white',pos=[0,-40])
    background.draw()
    responseText.draw()
    promptText.draw()
    #Create the text, then draw it.
    win.flip()
    timer.reset()
    #start the timer
    while True:
        response=event.waitKeys(keyList=['f','j'])[0]
        break
        #Wait for a response, and leave the loop when you have it
    rt = timer.getTime()
    responsewriter.writerow({'Response': response, 'RT': rt})
    #Write the row of data
```

Close up shop!

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

Final Result!

Citing

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