Events and Animation: The Bouncing Square

So far, the Tkinter's canvas widget has made drawing squares, circles, text, and even nyan Poptart cats, practical and relatively straightforward. Now its time to animate our canvas!

Global Vs. Local Canvas

We've now see two ways of accessing our canvas in the various functions that might need it: storing the widget's information within a global dictionary, or declaring the canvas widget itself as global. When displaying many objects on a canvas, however, we need a way to store the information of each object (perhaps its location or color) for easy access later. We'll use a global dictionary for our bouncing square example.

The init Function

When using a canvas in python, 5 lines need to be present nearly every time:

```
from Tkinter import *
root = Tk
canvas = Canvas(root, width = foo, height = bar)
canvas.pack()
root.mainloop()
```

How we organize these (and all subsequent lines) is highly flexible, however. In a complex animation, it's a good idea to have a run function handle these initializing steps, like so:

```
from Tkinter import *

def run():
        root = Tk
        canvas = Canvas(root, width = foo, height = bar)
        canvas.pack()
        root.mainloop()

run()
```

For our bouncing square example, there are quite a few variables we need to keep track of, including where our square is at any given moment, how large our square is, what color, etc. We'll create a global dictionary to keep track of all this data, and to change some as needed.

```
from Tkinter import *

def run():
        root = Tk
```

Events and Animation: The Bouncing Square

```
        foo = Canvas(root, width = foo, height = bar)
        canvas.pack()
        global data
        data = {}
        data["canvas"] = foo
        root.mainloop()

run()
```

This is not enough, however: with so many variables, it might be a good idea to have a function that initializes this type of information, so when we first draw the square (before it moves at all), everything exists. We call this an init function, and it might look like this:

```
from Tkinter import *

def init():
        global data
        data["squareColor"] = "blue"
        data["squareLeft"] = 50
        data["squareTop"] = 50
        data["squareSize"] = 50

def run():
        root = Tk
        foo = Canvas(root, width = foo, height = bar)
        foo.pack()
        global data
        data = {}
        data["canvas"] = foo
        init()
        root.mainloop()

run()
```

The redrawAll function

With our square's starting point established, it would be nice to see the square. To clear off the canvas of old images, we would need to call foo.delete(ALL), and then draw the square based off the data saved in the global dictionary. Remember that the syntax for drawing a square looks like this, where foo is the canvas, left/right/top/bottom are integers, and color is a string:

```
foo.create_rectangle(left, top, right, bottom, fill=color)
```

Events and Animation: The Bouncing Square

Timer Fired

Able to draw our square, we need to keep it moving across the canvas. A function that serves as a timer would be handy…perhaps using foo.after(delay, function) would help? Having the function, timerFired, change the data from which we draw our square, as well as having it call redrawAll, would be nice! An example, where doTimerFired is provided code that moves the square's data:

```
def timerFired():
    global data
    foo = data["foo"]
    doTimerFired()
    redrawAll()
    delay = 50
    foo.after(delay, timerFired)
```

Let's put it all together…

```
from Tkinter import *

def run():
    root = Tk()
    w = 600
    h = 500
    foo = Canvas(root, width=w, height=h)
    foo.pack()
    global data
    data = {}
    data["foo"] = foo
    data["w"] = w
    data["h"] = h
    init()
    timerFired()
    root.mainloop()

def init():
    global data
    data["squareColor"] = "blue"
    data["squareLeft"] = 50
```

Events and Animation: The Bouncing Square

```
        data[“squareTop”] = 50
        data[“squareSize”] = 50
        data[“counter”] = 0
        data[“goingR”] = True
        data[“goingD”] = False

def timerFired():
        global data
        foo = data[“foo”]
        doTimerFired()
        redrawAll()
        delay = 50
        foo.after(delay, timerFired)

def doTimerFired():
        global data
        if  data[“goingR”]:
                if (data[“squareLeft”] + data[“squareSize”] > data[”w”]):
                        data[“goingR”] = False
                else:
                        moveRight()
        else:
                if (data[“squareLeft”] < 0):
                        data[“goingR”] = True
                else:
                        moveLeft()
        if data[“goingD”]:
                if (data[“squareTop”] + data[“squareSize”] > data[”h”]):
                        data[“goingD”] = False
                else:
                        moveDown()
        else:
                if (data[“squareTop”] < 0):
                        data[“goingD”] = True
                else:
                        moveUp()

def moveLeft():
        global data
        data[“squareLeft”] -= 20

def moveRight():
        global data
        data[“squareLeft”] += 20

def moveUp():
        global data
        data[“squareTop”] -= 20

def moveDown():
        global data
        data[“squareTop”] += 20

def redrawAll():
        global data
        data[“foo”].delete(ALL)
        data[“foo”].create_rectangle(data[“squareLeft”],
                                     data[“squareTop”],
                                     data[“squareLeft”] + data[“squareSize”],
                                     data[“squareTop”] + data[“squareSize”],
                                     fill=data[“squareColor”])
run()
```