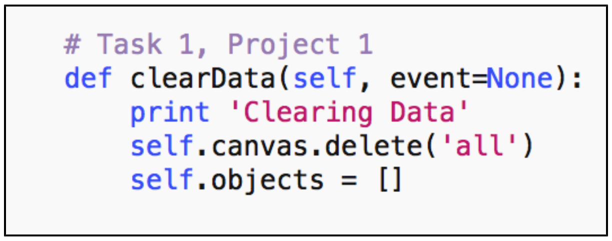
CS251 Proj1

CS251 Project 1 - Designing A GUI

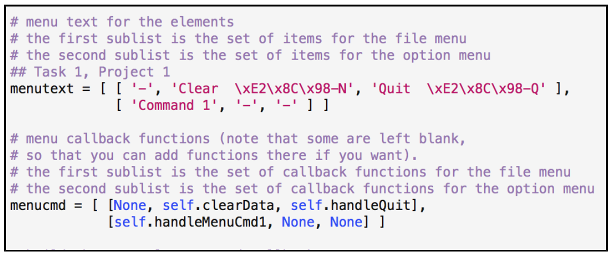
The purpose of this project was to start building an interactive data visualization system. To start this multi-week project, we have to become familiar with Tkinter GUI building system in Python. The backbone of our GUI project is a data visualization program that allows the user to insert data dots of different colors. At this point, the creation of points generates random points. The main tasks of this first project was to add menus, mouse callbacks, dialog window, and listboxes to our current visualization program.

Task 1. This task asked us to make a clearData method that would handle creating a keyboard short-cut for clearing all data point. Also, we are asked to bind this method to a new item in the file menu. First, I created the method for clearData and used the Tk canvas.delete function to clear the points. Here is a what my code for this task looked like:



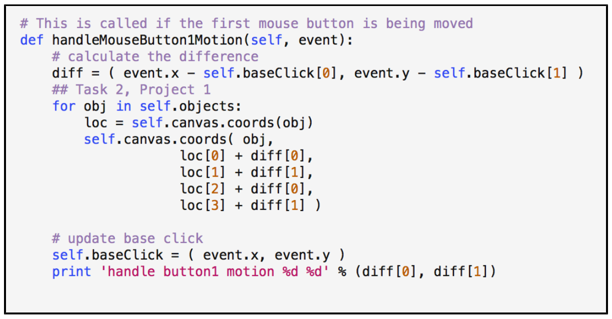
```
    # Task 1, Project 1
    def clearData(self, event=None):
        print 'Clearing Data'
        self.canvas.delete('all')
        self.objects = []
```

Next, we were tasks with binding this method the the keyboard shortcut Cmd-N and to a new item in the file menu. To do this, I edited the code associated with menutext and menucmd under the method buildMenus and added the "Clear" function as necessary. Here is my code snippet:

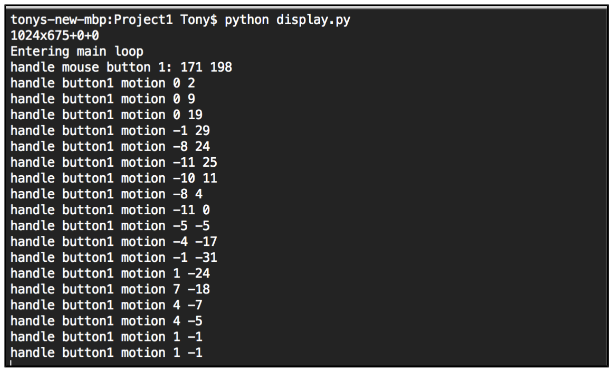


```
# menu text for the elements
# the first sublist is the set of items for the file menu
# the second sublist is the set of items for the option menu
## Task 1, Project 1
menutext = [ [ '-', 'Clear  \xE2\x8C\x98-N', 'Quit  \xE2\x8C\x98-Q' ],
             [ 'Command 1', '-', '-' ] ]

# menu callback functions (note that some are left blank,
# so that you can add functions there if you want).
# the first sublist is the set of callback functions for the file menu
# the second sublist is the set of callback functions for the option menu
menucmd = [ [None, self.clearData, self.handleQuit],
            [self.handleMenuCmd1, None, None] ]
```

Task 2. The goal of this second task was to two new functions to the mouse capabillities. The first goal was to allow the motion of mouse click 1 move the points and present the user with the (x,y) coordinates of how much the data points have moved as a whole. To this, we had to edit the handleMouseButton1Motion method. When the user clicks and moves the points around, the terminal will give him/her a constant print-out update of the movement of the data points. Here is a snippet of my code and the terminal print outs:



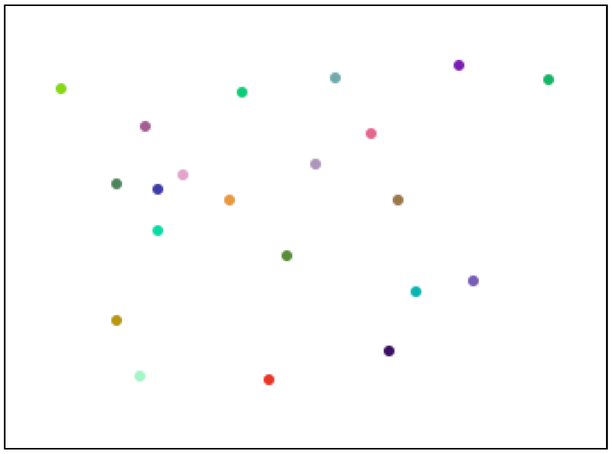
```
# This is called if the first mouse button is being moved
def handleMouseButton1Motion(self, event):
    # calculate the difference
    diff = ( event.x - self.baseClick[0], event.y - self.baseClick[1] )
    ## Task 2, Project 1
    for obj in self.objects:
        loc = self.canvas.coords(obj)
        self.canvas.coords( obj,
                            loc[0] + diff[0],
                            loc[1] + diff[1],
                            loc[2] + diff[0],
                            loc[3] + diff[1] )

    # update base click
    self.baseClick = ( event.x, event.y )
    print 'handle button1 motion %d %d' % (diff[0], diff[1])
```



```
tonys-new-mbp:Project1 Tony$ python display.py
1024x675+0+0
Entering main loop
handle mouse button 1: 171 198
handle button1 motion 0 2
handle button1 motion 0 9
handle button1 motion 0 19
handle button1 motion -1 29
handle button1 motion -8 24
handle button1 motion -11 25
handle button1 motion -10 11
handle button1 motion -8 4
handle button1 motion -11 0
handle button1 motion -5 -5
handle button1 motion -4 -17
handle button1 motion -1 -31
handle button1 motion 1 -24
handle button1 motion 7 -18
handle button1 motion 4 -7
handle button1 motion 4 -5
handle button1 motion 1 -1
handle button1 motion 1 -1
```

The second half of this task involved adding code to create a circular data object of random color at a mouse click location and store it in self.object list. I edited handleMouseButton3 for this. Here is my code and an example of random color points on the canvas:

```
## Task 2, Project 1
def handleMouseButton3(self, event):
    self.baseClick = (event.x, event.y)
    print 'handle mouse button 3: %d %d' % (event.x, event.y)

    dx = 3
    rgb = "#%02x%02x%02x" % (random.randint(0, 255),
                             random.randint(0, 255),
                             random.randint(0, 255) )
    oval = self.canvas.create_oval( event.x - dx,
                                    event.y - dx,
                                    event.x + dx,
                                    event.y + dx,
                                    fill = rgb,
                                    outline='')

    self.objects.append( oval )
```



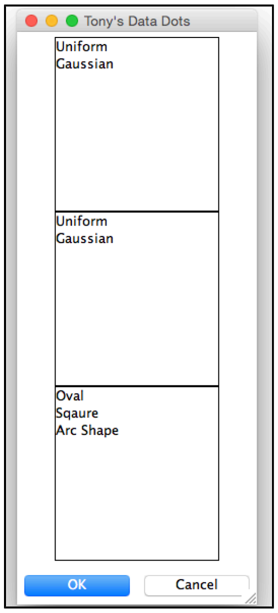
Task 3. This task was focused on creating Dialog ListBoxes to handle the type of random point generation that occurs when the user selects the "Generate Points" button. To this, we had to first copy over code from Effbot website on the standard Dialog support class. Then, we had to create dialog boxes that gave the user to choose the type of random distribution generated fro the X-coordinate and the Y-coordinate. I received some guidance and help from fellow CS251 classmate Julia Saul with some of Task 3. First, I create a ListBox class with two dialog boxes with the options "Normal" and "Gaussian" Here is that code:

```
## Task 3, Project 1
# Received help from cs251 classmate JSaul
class ListDialog(Dialog):

    def body(self, master):

        self.box1 = tk.Listbox(master, selectmode=tk.SINGLE, exportselection=0)
        self.box1.insert(tk.END, "Uniform")
        self.box1.insert(tk.END, "Gaussian")

        self.box2 = tk.Listbox(master, selectmode=tk.SINGLE, exportselection=0)
        self.box2.insert(tk.END, "Uniform")
        self.box2.insert(tk.END, "Gaussian")

        #Extension 2 continued...
        self.box3 = tk.Listbox(master, selectmode=tk.SINGLE, exportselection=0)
        self.box3.insert(tk.END, "Oval")
        self.box3.insert(tk.END, "Sqaure")
        self.box3.insert(tk.END, "Arc Shape")

        self.box1.pack(side=tk.TOP)
        self.box2.pack(side=tk.TOP)
        self.box3.pack(side=tk.TOP)

    def apply(self):
        self.result = [map(int,self.box1.curselection()),map(int,self.box2.curselection())
        pass #override
```

Then, in the generateRandomDataPoints method, I create a For loop with conditionals that handled each possible outcome of the list box choices. For example [0][0] = [0] means the user chose the first option in the first row and the first column. There should be four conditionals, one for each option Xnormal, Xgaussian, Ynormal, Ygaussian. Also, the ListBox will pop up when the user chooses to generate new data points. Here is the code:

```
for i in range(int(self.numOption.get())):
    if dialog.result[0][0] == 0:
        x = random.randrange(0,self.canvas.winfo_width())
    else:
        x = random.gauss(self.canvas.winfo_width()/6,self.canvas.winfo_width()/10)

    if dialog.result[1][0] == 0:
        y = random.randrange(0,self.canvas.winfo_height())
    else:
        y = random.gauss(self.canvas.winfo_width()/6,self.canvas.winfo_width()/10)
```

Here is what my ListBoxes look like:



*Extensions. *For this project, I did 3 extensions. You can see my second extension in the picture above.

Extension 1. For this extension, I create a new dropdown box on the right side of my visualization program next to the dropdown for choose color. This one was called numOption and it allowed the user to control the specific number of points generated when the make new random data points. To make this work, I just called the numOption.get() function as the number controlling my for loop in my generateRandoDataPoints method. Here is a snippet of my numOption dropbox code:

```
# EXTENSION 1
# make a new menubutton for numberOption
# this dropdown menu allows you to control how many dots there are
self.numOption = tk.StringVar( self.root )
self.numOption.set("200")
numMenu = tk.OptionMenu( rightcntlframe, self.numOption,
                                "200", "150", "100", "50", "25", "10" )
numMenu.pack(side=tk.TOP)       # EXTENSION 1
```

Extension 2. You can see this extension in the listbox picture above. Basically, I just made a third ListBox and allowed the user to choose the shape of the data point they are generating. Using canvas.create_sqaure,oval, and arc. First, I create a third ListBox in my ListBox class. Then, I update the createRandomDataPoints method inside the for loop for numOption with 3 new conditionals about the row and column [2][0] and what each choice does. Here is the code:

```
## EXTENSION 2
# this gives the user the option to choose the shape of the data point
# you can choose a oval, square, or arc shape
if dialog.result[2][0] == 0:
    pt = self.canvas.create_oval( x-dx, y-dx, x+dx, y+dx,
                        fill=self.colorOption.get(), outline='' )
    self.objects.append(pt)
elif dialog.result[2][0] == 1:
    pt = self.canvas.create_rectangle( x-dx, y-dx, x+dx, y+dx,
                        fill=self.colorOption.get(), outline='' )
    self.objects.append(pt)
else:
    pt = self.canvas.create_arc( x-dx, y-dx, x+dx, y+dx,
                        fill=self.colorOption.get(), outline='' )
    self.objects.append(pt)
```

Extension 3. This extension is similar to #1, but for this one I allowed the use to control the relative size of the shaped data point drawn. I created another dropdown menu box called sizeOption. This was just another dropdown box next to number and color options allowing the user to control the size of the data point. Here is the code:

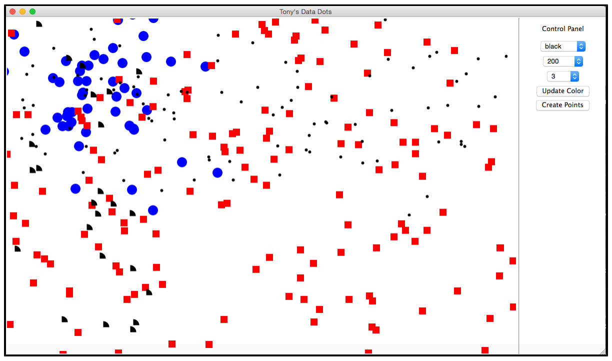
```
# EXTENSION 3!
# make a new menubutton for sizeOption
# this dropdown menu allows you to chose the size of the da
self.sizeOption = tk.StringVar( self.root )
self.sizeOption.set("3")
sizeMenu = tk.OptionMenu( rightcntlframe, self.sizeOption,
                        "12", "10", "7", "5", "3" )
sizeMenu.pack(side=tk.TOP)
```

Finally, to make this happen I had to update the createRandomDataPoints method. I simply changed the dx and dy that control how far each point of the shape was away from the data point. I simply wrote these two lines of code in that method:

```
# replace dx and dy with sizeOption to adjust size of data dx = int(self.sizeOption.get()) dy = int(self.sizeOption.get())
```

```
dx = int(self.sizeOption.get())
dy = int(self.sizeOption.get())
```

FINALLY, here is a picture of my program with a variation of the buttons, menus, and extensions I created:



What I learned. The goal of this lab was to work on building a data visualization program. Working on this project, I became familiar with Tkinter GUI building system in Python. I became comfortable working with GUI systems, mouse clicks and dialog boxes. I learned how to create menus, mouse callbacks, dialog window, and listboxes in a visualization program.

Who helped me. I worked alongside fellow CS251 classmates Brendan Doyle and Steven Parrot. I received help from another classmate, Julia Saul for Task 3.