At this point in time there are three array; aCount holds the data, aColor holds colors for each data category, and aCategory holds the labels for the data.

nTotal is a number that is the sum of all of the data provided. nElements is the total number of items in the array, in this example there are five elements. The counter, i, goes from 0 to <nElements because arrays begin with an index of zero. So i counts 0,1,2,3,4.

To calculate a percentage you need to divide the piece of data by the total of all of the data. This value is rounded to a whole number. After all of the percentages are calculated they are totaled. It is possible to come up with a total of 101 or 99 from rounding errors.

If this occurs we simple adjust the last percentage so that the sum will add to 100.

X=110 tells us how far across the screen to put the square color code symbol. i*20+8 calculated the vertical placement.

Color is an object. Color.setRGB(hexidecimal value) Allows the color of a movie clip to be set to a hexidecimal value.

To create a color object the syntax is – new Color(movieClip)

eval is necessary to turn the string into the object.

Since a circle has 360 degrees and the percentages add up to 100% each slice is rotated exactly 3.6 degrees.
This is the actionscript that is used to place the variables into arrays.

```actionscript
onClipEvent(load) {
    this.loadVariables("arrayinfo.txt");
}
```

```actionscript
onClipEvent(data) {
    // Once data is read in, convert it to arrays
    aCount = countValues.split(",");
    aColor = colorValues.split(",");
    aCategory = categoryValues.split(",");

    // Change aCount to number values instead of strings
    for (var i in aCount) {
        aCount[i] = Number(aCount[i]);
    }

    // Change color values from hex strings to numbers
    for (var i in aColor) {
        aColor[i] = parseInt(aColor[i], 16);
    }

    // Copy all the variables into the main timeline (so the
    // actions in frame 5 will not need to be modified)
    for (item in this) {
        //trace(this[item]);
        _root[item] = this[item];
    }

    // Continue with movie
    _root.gotoAndStop("draw");
}
```

This is the text file that is used to supply data to the pie chart.

```text
countValues=20,10,40,5,25&colorValues=00ffff,ff0000,00ff00,0000ff,ffff00&categoryValues=dogs,cats,fish,hamsters,birds
```

The structure of this data is very important to understand. It is delivered via name/value pairs. Each variable is separated by the ampersand (&). Each variable consists of a name and values. The name is followed by an equals sign. The values are separated by commas. This is a standard format for a data string and it is used in cgi scripts and in many web applications.

When the controller clip is loaded into the first frame this triggers a ClipEvent. When the ClipEvent occurs the script is executed. The first command is to load variables from a text file into the movieclip instance. The text file is called arrayinfo.txt

```actionscript
onClipEvent(data) - this event is triggered when the movie clip completely receives data via loadVariables or loadMovie
```

The split command takes a string and used a given symbol, in this case a comma, to separate the string into elements in an array. The delimiter (the comma) is not placed into the array.

The format for the parseInt command is

```javascript
parseInt(expression,radix)
```

In this case an expression is a text string such as 00ff00 and the sixteen means that this command should be converted to base 16 or hexadecimal. This is necessary is you are going to specify a color.

Remember that all of the variables were loaded into a movie clip called the controller. Now that they have been manipulated into a useable format they are moved to the root of the Flash movie. Note that the trace command has been commented out – which means it was used when the movie was being built and then removed by putting // in front of it.

Once the variable are ready it is time to go to the draw frame.