

## *Arrays Again - Coding a Graph: (These notes use the class code set 13)*

*We have given two reasons for using arrays:*

- we can store a lot of data under a single variable stores the individual values*
- we can reduce the size of your code by using loops with the variable that is storing the data.*

*In this set of notes, we will use an array of gold prices<sup>1</sup> as data for drawing a graph. This is being done as an exercise to demonstrate new ideas and reinforce older concepts. The reinforcements include a step-wise development of the code starting with the “simpler” tasks and moving into the more complex - ideally learning along the way. Other ideas are global variables, the use of the map() function, ifs and loops.*

*The new ideas concern the use of an array.*

*Jim has arbitrarily divided the program into a set of tasks. This is not presented as the best or the only way to develop the code. It is just one way. Feel free to find fault with his strategy. The more you think about this, the better prepared you will be for your eighth homework and the projects.*

*Here are the steps Jim used to develop the code:*

- 1. Get the data and code the array declaration and the initialization:*

```
int [ ] gold = { 185, 133, 139, 180, 250, 616, 500, 366, 464, 395,  
                284, 352, 405, 432, 488, 415, 358, 350, 329, 376,  
                377, 398, 353, 292, 286, 294, 260, 292, 257, 400,  
                434, 564, 683, 937, 978, 1094, 1401 };
```

- 2. Add some variables to define the edges of the graph. This will make the adjustments of the final graph easier.*

```
float leftEdge, rightEdge, topEdge, bottomEdge;
```

```
void setup( )  
{
```

---

<sup>1</sup> This is the closing price of gold in dollars on the London market on or about February 23 from 1975 through this year.

```

size( 1000, 600 );
f = loadFont( "f.vlw" );
textFont( f );
textSize( 12 );
textAlign(CENTER, CENTER);
rectMode( CORNERS );

leftEdge = 10;
rightEdge = width-10;
topEdge = 20;
bottomEdge = height-20;
}

```

3. Plot points horizontally based on the years of the data. This ignores the actual values of the gold prices in the array. We just want to be sure that we can accurately space the years across the window. This code is in the **set13A** folder

```

void testPlots( )
{
  fill( 0 );
  stroke( 0, 0, 255 );
  strokeWeight( 4 );
  for ( int i = 0; i < gold.length; i++)
  {
    float x =
      map ( i,
           0, gold.length,
           leftEdge, rightEdge);
    float y = height/2;

    point( x, y );
  }
}

```

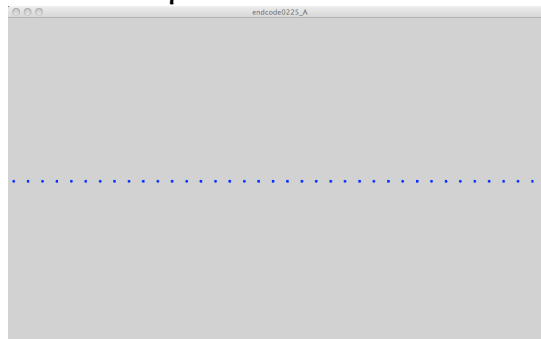
We use a for loop to traverse the array.  
 For each iteration we compute the x value of the point using the map function:

- < Map this value
- < which is between these two values
- < into a range between these two values.

We do not care about the y value - we just want to prove that we can find the correct place for the years

- < Draw the point

*This code gives us this output:*

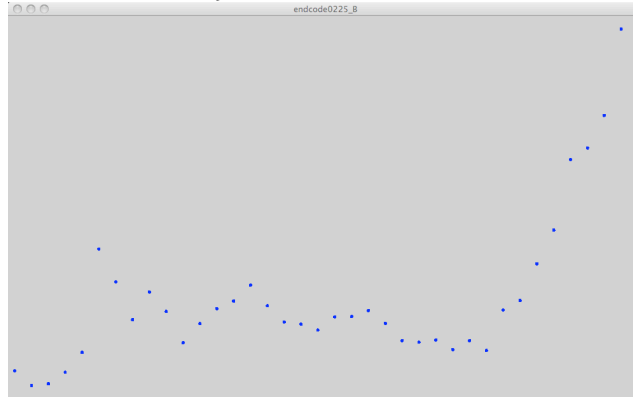


3. Now that we know we can plot the horizontal distances for each year, we can concentrate on the vertical location of each point using the data in the array.

This code is in the **set13B** folder

<pre> int minValue, maxValue;  void setup( ) {   . . .   minValue = min(gold);   maxValue = max(gold); }  void testPlots( ) {   fill( 0 );   stroke( 0, 0, 255 );   strokeWeight( 4 );    for ( int i = 0; i &lt; gold.length; i++)   {     float x =       map( i,           0, gold.length,           leftEdge, rightEdge);      float y =       map( gold[i],           maxValue, minValue,           topEdge, bottomEdge);      point( x, y );   } </pre>	<p>To compute the horizontal position of the points using the data in the graph, we will use the map function to map each value in the array to the vertical location in the graph. To use the map function we must know the max and min values in the array. We need two new global variables:</p> <p>&lt;</p> <p>And we have to determine their values. We do this the setup( ) function using the &lt; min( ) and the &lt; max( ) functions.</p> <p>We go back inside the for loop that is traversing the array. Old code from the previous part</p> <p>For each iteration we compute the y value of the point using the map function with the data in the array:</p> <ul style="list-style-type: none"> <li>&lt; Map the value of element [ i ] of gold</li> <li>&lt; which is between these two values</li> <li>&lt; into a range between these two values.</li> </ul> <p>&lt; Draw the point.</p>
---	---

*This code gives us this output:*



4. We can successfully locate points for the data in the array horizontally and vertically. Now we will label each point with the year. Initial tests showed that the four-digit year is so long it overlaps the neighboring values so we will use only two digits or at least we try to do that. This attempt will use the % operator to strip away the first two digits.

*This code is in the **set13C** folder*

```
final int FIRST_YEAR = 1975;
final int LAST_YEAR = 2011;

void testPlots( )
{
  fill( 0 );
  stroke( 0, 0, 255 );
  strokeWeight( 4 );

  for ( int i = 0; i < gold.length; i++)
  {
    float x = map( . . . );
    float y = map( . . . );
    point( x, y );
    int year =
      (i + FIRST_YEAR)%100;
    text( year, x, y-10 );
  }
}
```

We are still inside the for loop that is traversing the array. The loop iterates from 0 to 36 for the 37 values in the array. We need to display the years 75 to 11 for 1975 to 2011. To make this process easier, we add two new global constants. These make the code more readable and, if we want to expand the data to include years before 1975 or revisit this next year, we can keep the graph accurate by just editing the values of these constants

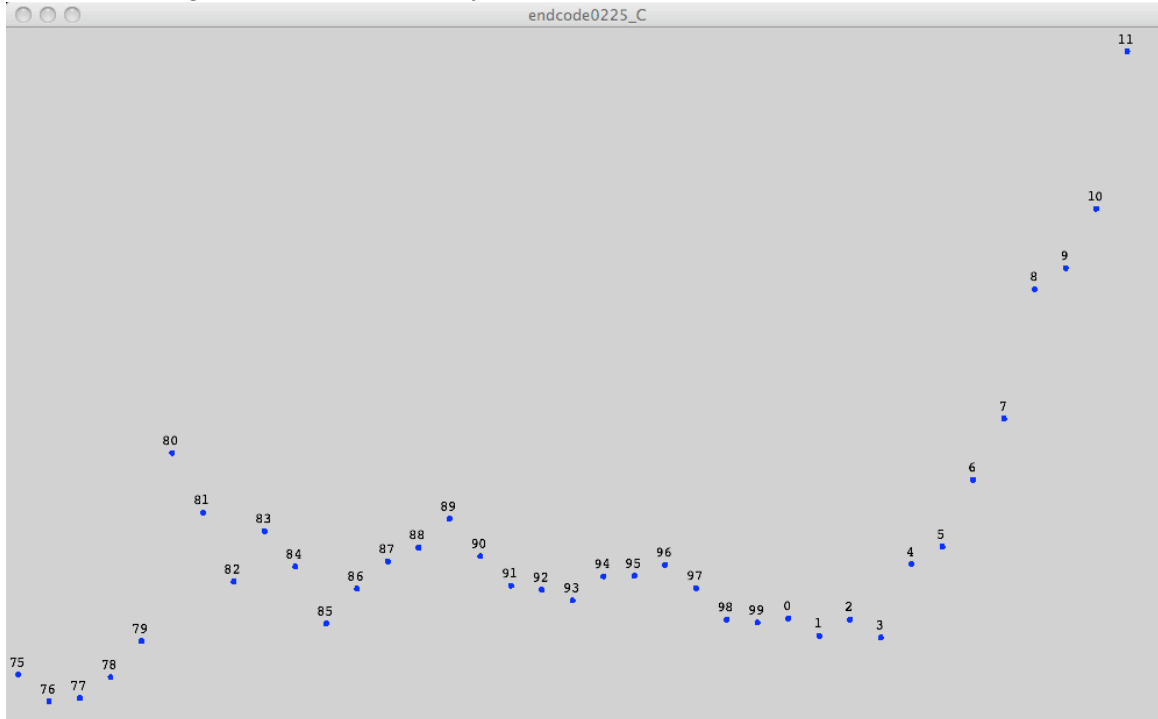
*Refer to the previous parts for explanation on this code.*

We compute the value to be displayed and store it in a local variable named year. The arithmetic we use adds 1975 to the for loop variable. When i is zero, this addition gives us the value 1975. To "strip" away the first

two digits, we use the mod operator ( the % ) for division. This evaluates to the last two digits of the year.

$$\begin{array}{r}
 19 \text{ R } 75 \\
 \hline
 1975 \% 100 \rightarrow 100 \text{ ) } 1975 \\
 \underline{-100} \\
 975 \\
 \underline{-900} \\
 75
 \end{array}$$

*This code gives us this output:*



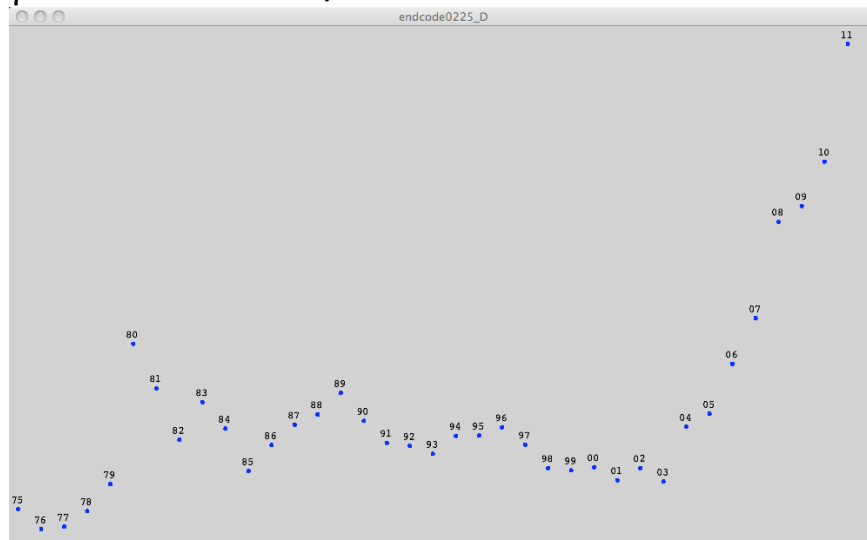
5. It is difficult to see in the screen print but the years 2000 to 2009 have only one digit. The leading zero is suppressed. There are several ways to fix this<sup>2</sup> and Jim decided not to introduce any new syntax or functions at this point. So his “fix” will use an **if/else** control structure as we will see.

<sup>2</sup> The functions **nf( )**, **nfc( )**, **nfp( )**, and **nfs( )** can be used to format output with the **text( )** function. The API explains these nicely and they are simpler to use than the code that Jim is showing here.

*This code is in the **set13D** folder*

<pre> void testPlots( ) {   fill( 0 );   stroke( 0, 0, 255 );   strokeWeight( 4 );    for ( int i = 0; i &lt; gold.length; i++)   {     float x = map( . . . );     float y = map( . . . );     point( x, y );      // label the points:     int year =       (i + FIRST_YEAR)%100;     // add a leading zero for years     // 2000 to 2009     if ( year &lt; 10 )     {       text( "0" + year, x, y-10 );     }     else     {       text( year, x, y-10 );     }     // end new code   } } </pre>	<p>We are still inside the for loop that is traversing the array. The strategy is to "look" at the value of the year variable that we want to print. If it is less than 10, we add a zero to the first argument of the call of the function text( )</p> <ul style="list-style-type: none"> <li>&lt; Test the value of year</li> <li>&lt; Here we add the zero to the text output.</li> <li>&lt; Here we do not.</li> </ul>
---	--

*This output is the result of the new code:*



*Again, it is a bit difficult to see but it works.*

6. Now we will add code to connect the points that we have plotted. This code is in the **set13E** folder:

```
void testPlots( )
```

```
{
```

```
  fill( 0 );
```

```
  float oldX, oldY;
```

```
  oldX = 0;
```

```
  oldY = 0;
```

```
  for ( int i = 0; i < gold.length; i++)
```

```
  {
```

```
    float x = map( . . . );
```

```
    float y = map( . . . );
```

```
    stroke( 0, 0, 255 );
```

```
    strokeWeight( 4 );
```

```
    point( x, y );
```

```
    int year =
```

```
      (i + FIRST_YEAR)%100;
```

```
    if ( year < 10 )
```

```
    {
```

```
      text( "0" + year, x, y-10 );
```

```
    }
```

```
    else
```

```
    {
```

```
      text( year, x, y-10 );
```

```
    }
```

```
    // connect points with lines
```

```
    if ( i > 0 )
```

```
    {
```

```
      strokeWeight( 1 );
```

```
      stroke( 0 );
```

```
      line( oldX, oldY, x, y );
```

```
    }
```

```
    // save current value of x and y
```

```
    // for use in the next iteration
```

```
    oldX = x;
```

```
    oldY = y;
```

```
  }
```

```
}
```

The way we will connect the points is to declare two local variables named `oldX` and `oldY`. We will use these to "remember" the (x, y) values of the previous point. This is more efficient than computing them again. Since these are local variables, we must initialize them which is done here:

```
<
```

```
<
```

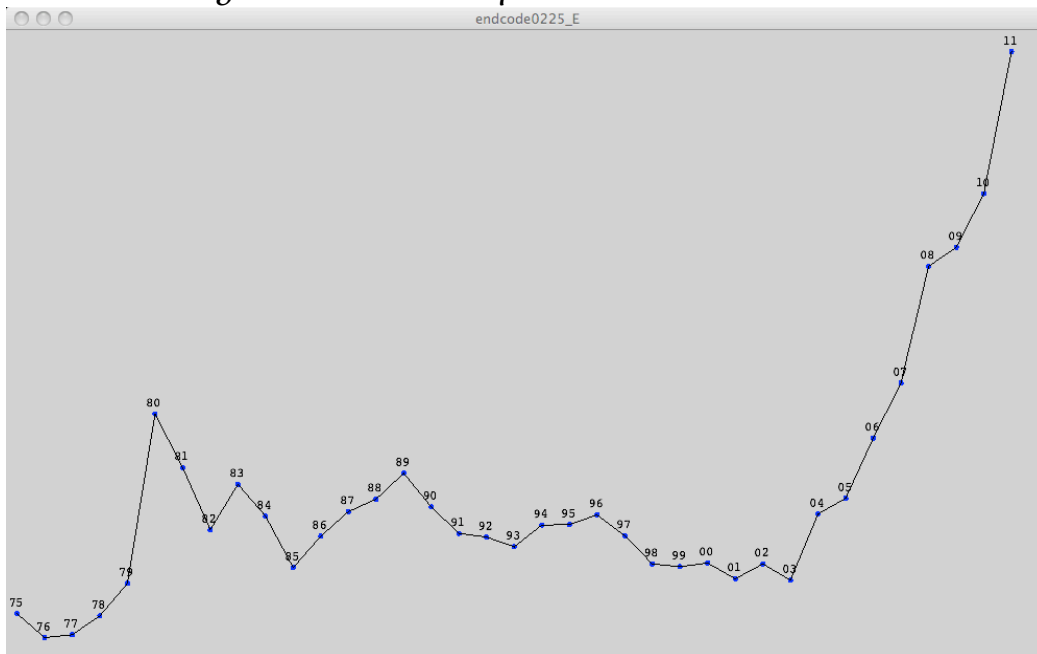
We are still inside the for loop that is traversing the array.

We have to move these two lines of code down into the loop because we are going to alter these values to draw the lines that connect the points.

We will connect this point to the previous point. The point for element [0] has no previous point so we need to use an if to insure that we do not try to do this. Otherwise, the code crashes with an `arrayindexoutofbounds` error.

Finally we "remember" the current value of x and y for the point so we can use it to draw the line in the next iteration.

This new code gives us this output:



7. Next, we will color the lines. Lines that show an increase of the price will be **green** and lines that show a decrease will be **red**. This code is in the **set13F** folder:

```
void testPlots( )
{
  fill( 0 );
  float oldX, oldY;
  oldX = 0;
  oldY = 0;
  for ( int i = 0; i < gold.length; i++)
  {
    float x = map( . . . );
    float y = map( . . . );
    stroke( 0, 0, 255 );
    strokeWeight( 4 );
    point( x, y );
    fill( 0 );
    int year =
      (i + FIRST_YEAR)%100;
    if ( year < 10 )
    {
      text( "0" + year, x, y-10 );
    }
    else
    {

```

We are still working inside the for loop that is traversing the array.



```

    text( year, x, y-10 );
  }
  // for years after 1975,
  // compare current price to last
  // year's price and color line
  // green for gains and red for
  // losses
  if ( i > 0 )
  {

    if ( y < oldY )
    {
      stroke( 0, 200, 0 );
    }
    else
    {
      stroke( 255, 0, 0 );
    }
    strokeWeight( 1 );
    line( oldX, oldY, x, y );
  }
  oldX = x;
  oldY = y;
}

```

From the previous code, remember that we are drawing lines from this point back to the previous point. The [0]th point has no previous point so we use an if to avoid trying to draw a line from the [0]th point to the [-1] point.

<

We could compare the values of gold in the array but we used those values to compute the y location of the point so we can use the y values. There can be a bit of confusion here. We mapped the y value to the price of gold in a way that gives us smaller values for y when the price of gold is higher.

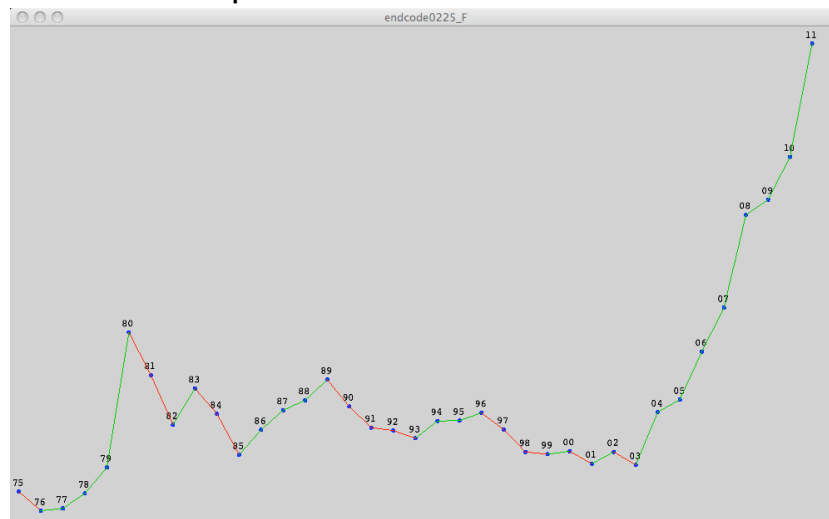
So if the value of y is smaller than the old value of y, the price of gold is higher. In this case, we color the line green.

< This is green.

Otherwise we color the line red.

< This is red

Here are the results of the new code:



8. The next task is to draw some gold bars - hey . . it is the price of gold. This code is in the **set13G** folder:

```
void setup( )
{
  size( 1000, 600 );
  f = loadFont( "f.vlw" );
  textFont( f );
  textSize( 12 );
  textAlign(CENTER, CENTER);
  rectMode( CORNERS );

  void testPlots( )
  {
    fill( 0 );
    float oldX, oldY;
    oldX = 0;
    oldY = 0;
    for ( int i = 0; i < gold.length; i++)
    {
      float x = map( . . . );
      float y = map( . . . );
      stroke( 0, 0, 255 );
      strokeWeight( 4 );
      point( x, y );
      fill( 0 );
      int year =
        (i + FIRST_YEAR)%100;
      if ( year < 10 )
      {
        text( "0" + year, x, y-10 );
      }
      else
      {
        text( year, x, y-10 );
      }
      if ( i > 0 )
      {
        if ( y < oldY )
        {
          stroke( 0, 200, 0 );
        }
        else
        {
          stroke( 255, 0, 0 );
        }
      }
    }
  }
}
```

To simplify the drawing of the gold bars we will use a `rectMode` argument named: **CORNERS**

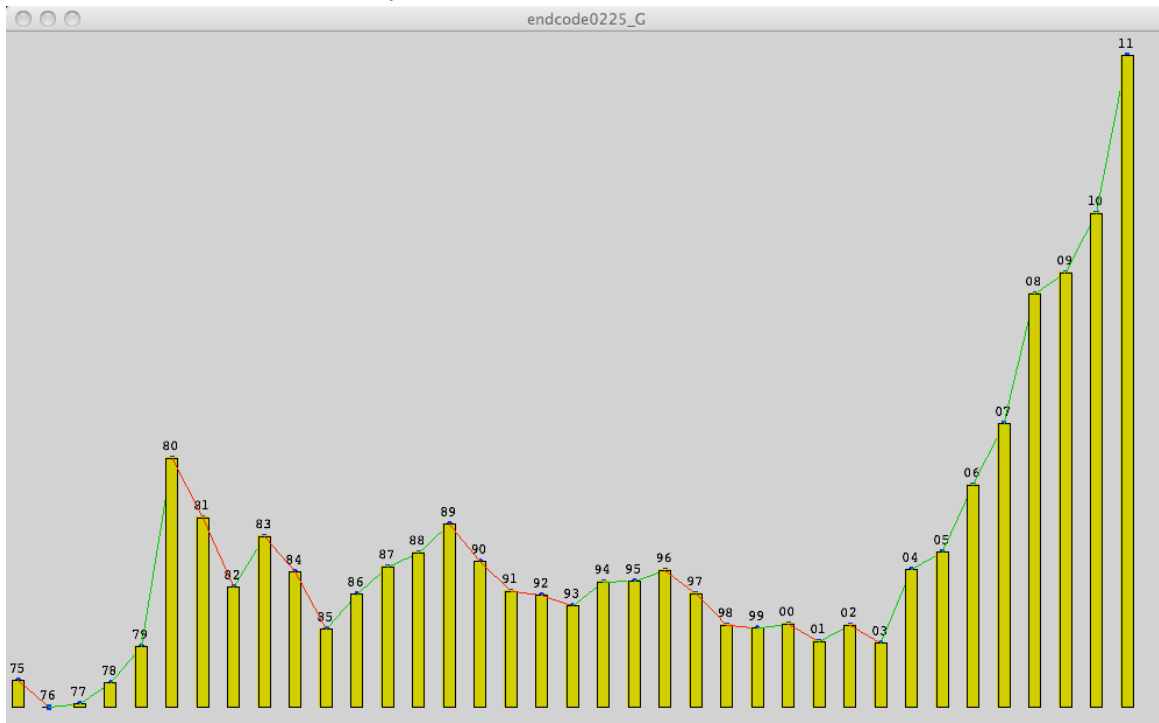
This allows us to specify the upper left corner and the lower right corner of the rectangle. We set this mode in the `setup` function:

<

We are still working inside the `for` loop that is traversing the array.

<pre> strokeWeight( 1 ); line( oldX, oldY, x, y ); } // draw the gold bars fill( 200, 200, 0 ); stroke(0); strokeWeight( 1 );  rect   ( x-5, y,     x+5, bottomEdge );  oldX = x; oldY = y; } } </pre>	<p>We do not need any new variables to do this since we have computed the x and y value of the point. We have a variable for the bottom edge of the graph.</p> <ul style="list-style-type: none"> <li>&lt; We set the fill to a sorta' gold color.</li> <li>&lt; We set the stroke to black and</li> <li>&lt; the stroke width to 1 pixel.</li> </ul> <p>We draw a rect</p> <ul style="list-style-type: none"> <li>&lt; This is the top left corner and</li> <li>&lt; this is the bottom right corner.</li> </ul> <p>We use -5 and +5 to give us a bar width of 10 pixels. The variable y will be the top of the bar at the level of the point. The variable, bottomEdge will be the bottom of the bar.</p>
--	---

Here are the results of the new code:



9. We will compute the mean value of the data in the array.  
This code is in the **set13H** folder:

```
int [ ] gold = { . . . };
PFont f;

final int FIRST_YEAR = 1975;
final int LAST_YEAR = 2011;

float leftEdge, rightEdge, topEdge,
bottomEdge;

int minValue, maxValue;

float mean;

void setup( )
{
  size( 1000, 600 );
  . . .
  mean = computeMean( );
  println( mean );
}

void draw( )
{
  testPlots( );
  noLoop( );
}

float computeMean( )
{
  float sum = 0;
  for(int i = 0; i < gold.length; i++)
  {
    sum += gold[i];
  }
  float mean = sum/gold.length;

  return mean;
}
```

We add a new global variable to store the mean value.

<

We call a new function - defined below - and assign the value it returns to the new global variable mean.

<

< We print it in the console window to check its value.

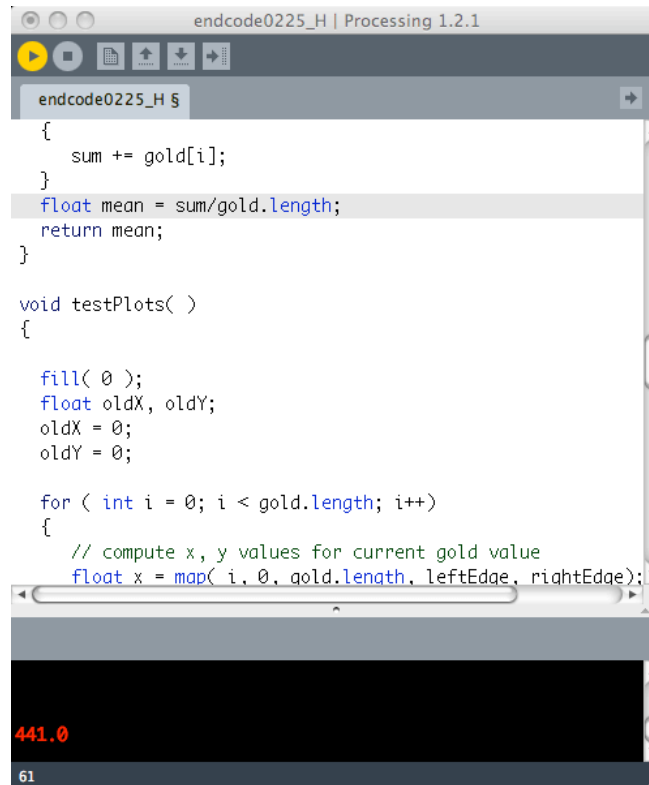
Here is the definition of the computeMean( ) function.

We compute the sum using the strategy discussed in class on Monday and Friday.

< We divide the sum by the length of the gold array and

< return the result

Since we are not altering the graphing code, the graphics window remains unchanged. However, the value of the mean is displayed in the console window:



```
endcode0225_H | Processing 1.2.1
endcode0225_H $
{
  sum += gold[i];
}
float mean = sum/gold.length;
return mean;
}

void testPlots( )
{
  fill( 0 );
  float oldX, oldY;
  oldX = 0;
  oldY = 0;

  for ( int i = 0; i < gold.length; i++)
  {
    // compute x, y values for current gold value
    float x = map( i, 0, gold.length, leftEdge, rightEdge);
```

441.0

61

10. Using the computed mean value, we will draw a horizontal line to mark the value on the graph. This code is in the **set13I** folder:

```
void draw( )
```

```
{
  testPlots( );

  plotMean( );

  noLoop( );
}
```

```
void plotMean ( )
```

```
{
  float y =
    map( mean,
        maxValue, minValue,
        topEdge, bottomEdge);
  stroke( 0, 0, 255 );
  strokeWeight( 2 );
  line( leftEdge, y, rightEdge, y );
  fill( 0, 0, 255 );
  text( "Mean", leftEdge+20, y-10 );
}
```

Since we do not need a for loop to plot the mean line on the graph, we will use a new function to draw the line.

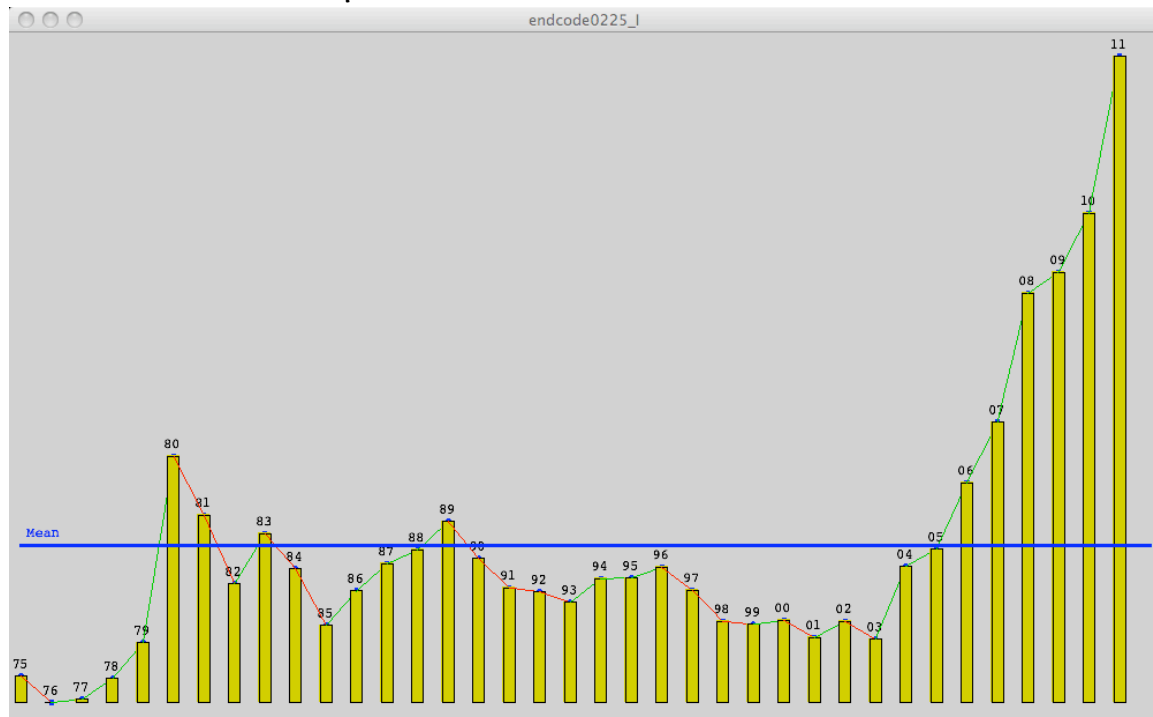
<

Here is the definition of the function, plotMean( ).

We compute the y coordinate of the mean line using the map function to map the mean into the vertical size of the graph.

< We draw a line at the computed y location and  
< we label it.

Here is the result of the new code:



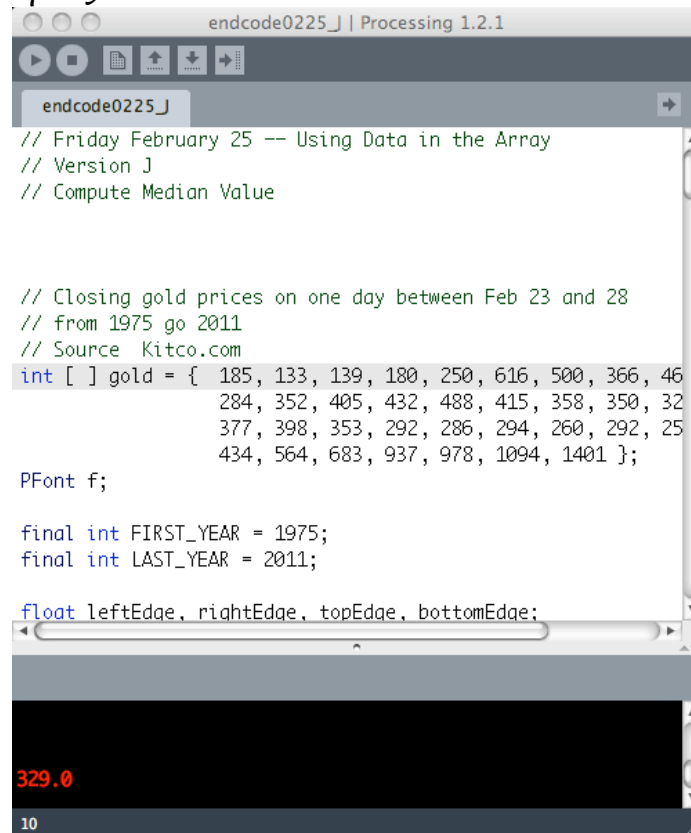
11. We will compute the median value of the data in the array in a manner very similar to the way we computed the mean.

This code is in the **set13J** folder:

<pre>int [ ] gold = { . . . }; PFont f; final int FIRST_YEAR = 1975; final int LAST_YEAR = 2011; float leftEdge, rightEdge, topEdge, bottomEdge; int minValue, maxValue; float mean, median;  void setup( ) {   size( 1000, 600 );   . . .   mean = computeMean( );   median = computeMedian( );   println(median); }  . . .  float computeMedian( ) {   int [ ] sortedGold = sort( gold );    int medianIndex = gold.length/2;    float median = gold[ medianIndex ];   return median; }</pre>	<p>We add a new global variable to store the median value.</p> <p>&lt;</p> <p>We call a new function - defined below - and assign the returned value to the new global variable median.</p> <p>&lt;</p> <p>&lt; We print it in the console window to check its value.</p> <p>Here is the definition of the computeMedian( ) function. This code uses a Processing API function named <b>sort( )</b>. This function requires an array as the argument. It returns a new array containing the original values of the argument but in sorted order with the smallest value in the [0]th element.</p> <p>The median value is the middle value or the value that is the middle element. The index of the middle element is <code>gold.length/2</code> when we have an odd number of values.<sup>3</sup> We are lucky. The array has an odd number of elements.</p> <p>&lt; We find the median value and &lt; return it.</p>
---	--

<sup>3</sup> If there are an even number of values, we are supposed to average the two middle values.

*Since we are not altering the graphing code the graphics window remains unchanged. However, the value of the median is displayed in the console window:*

A screenshot of a Processing IDE window titled "endcode0225\_J | Processing 1.2.1". The window is split into two panes. The top pane shows code for calculating the median of an array of gold prices. The code includes comments, variable declarations, and an array definition. The bottom pane is a black console window displaying the output "329.0" in red text. The status bar at the bottom of the IDE shows the number "10".

```
endcode0225_J | Processing 1.2.1
endcode0225_J
// Friday February 25 -- Using Data in the Array
// Version J
// Compute Median Value

// Closing gold prices on one day between Feb 23 and 28
// from 1975 to 2011
// Source Kitco.com
int [ ] gold = { 185, 133, 139, 180, 250, 616, 500, 366, 46
                284, 352, 405, 432, 488, 415, 358, 350, 32
                377, 398, 353, 292, 286, 294, 260, 292, 25
                434, 564, 683, 937, 978, 1094, 1401 };

PFont f;

final int FIRST_YEAR = 1975;
final int LAST_YEAR = 2011;

float leftEdge, rightEdge, topEdge, bottomEdge;

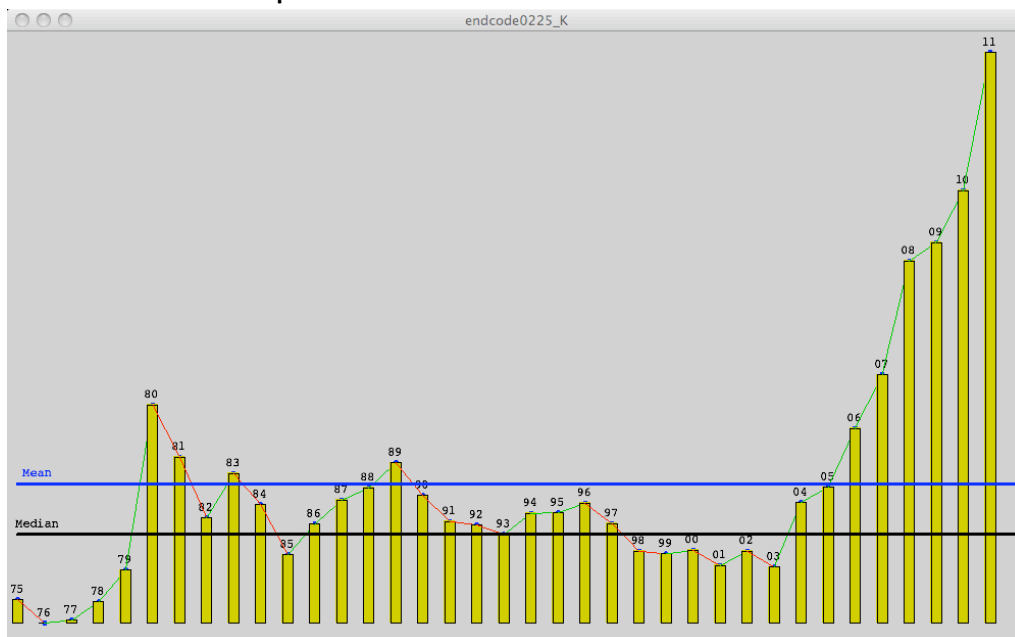
329.0
10
```



12. Using the computed median value, we will draw a horizontal line to mark the value on the graph using the same strategy we used to plot the mean line. This code is in the **set13K** folder:

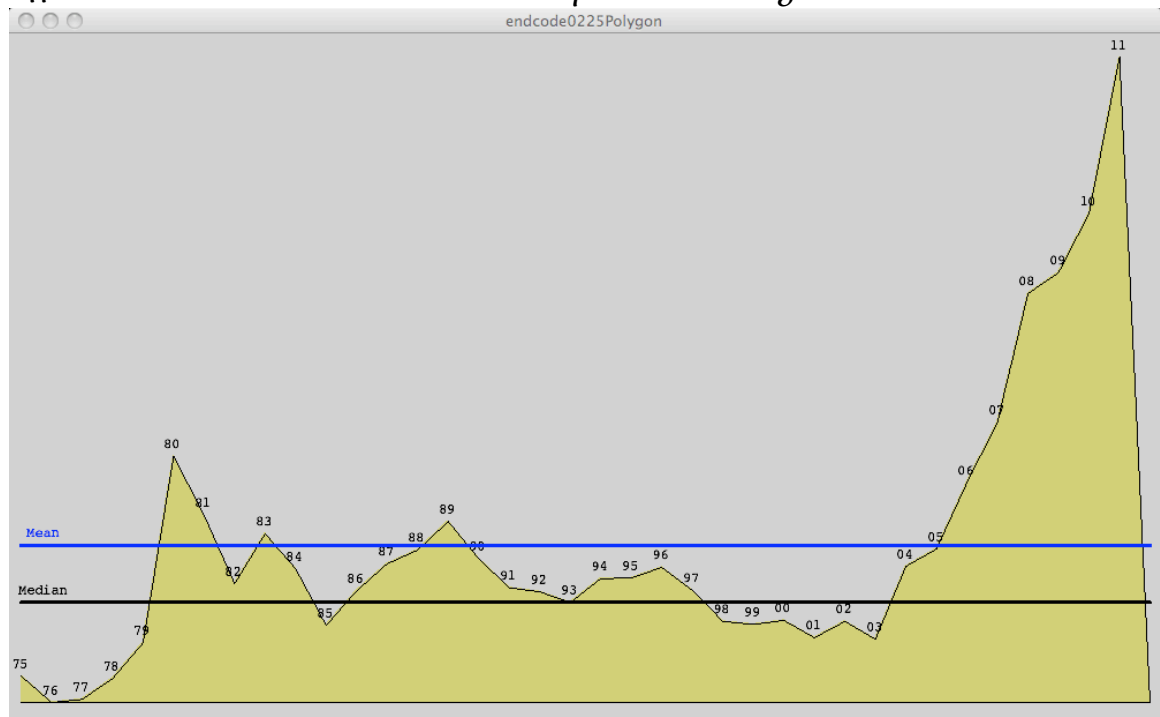
<pre>void draw( ) {   testPlots( );   plotMean( );    plotMedian( );   noLoop( ); }  void plotMedian( ) {   float y =     map( median,         maxValue, minValue,         topEdge, bottomEdge);   stroke( 0 );   strokeWeight( 2 );   line( leftEdge, y, rightEdge, y );   fill( 0 );   text( "Median", leftEdge+20, y-10 ); }</pre>	<p>Since we do not need a for loop to plot the median line on the graph, we will use a new function to draw the line.</p> <p>&lt;</p> <p>Here is the definition of the function, plotMedian( ).</p> <p>We compute the y coordinate of the median line using the map function to map the median into the vertical size of the graph.</p> <p>&lt; We draw a line at the computed y location and &lt; we label it</p>
---	--

Here is the result of the new code:



There is a lot here. If you work through this carefully, you should experience a learning curve in your understanding of arrays and how to use them. Refer to the class code that is copied into this set of notes. Try to alter it to do slightly different things. Here is a possible list:

1. Replace the bars with a circle that has a diameter mapped to the value of gold.
2. Replace the lines with the `beginShape( ) - vertex( )`, `endShape( )` functions. Here is what Jim ended up getting for his efforts... This will take some experimenting.



3. Using the `rotate( int )` function, try to wrap the bars around a point like a star. You will need to use the `translate( int, int )` function move to the middle of the screen. This can be done in 2-d space.
4. Add the horizontal and vertical axes lines, Put marks for the years and for \$ amounts and label them. You may have to alter the edge variable values to get these to fit on the window.

The more you work with arrays, the easier it will be to use them and to think about them as aids to achieving your goals in your code for homework 8 and the two projects.