Processing Notes

Chapter 9: Simple Functions

Often in programming we want to do a set of instructions multiple times. Lets say this set of instructions includes 10 statements. We could just put these 10 statements every place we use them, but if we use them many times, say 10, we end up writing lots of identical code for no reason. All that redundancy makes it hard to make changes, i.e. if you decide to change one of the statments you have to find that statement in 10 different places, and makes it easy to introduce errors, i.e. you might make a typo in some of the places and get it right in others, so the code become inconsistent.

Functions provide a clean/elegant way to reuse code without these problems. Functions should have a single logical task. Examples might include: "move the ball to the left", "draw a figure on the screen at location x,y", "enroll the student in class number COMP 1671", "calculate the student gpa", "give all employees a raise", or "calculate the mean temperature over the past year at a given longitude/latitude".

In fact, you have already been using functions: setup( ), draw( ), and mousePressed( ) are all functions. These are special functions that are part of the Processing model, but they are still just Java functions. A function has the general format:

returnType   functionName(parameter-list)
{

        function code

}

where returnType is the type of value returned by the function. If no value is returned that is specified with "void" as the return type. Note setup( ) , draw( ), and mousePressed( ) all have a "void" return type, i.e. they do not return a value. "functionName" is the name of the function, for example "setup", "draw" and "mousePressed" are all function names. The parameter list is the list of values, and their types, passed into a function. The functions setup( ), draw( ), and mousePressed( ) do not have any parameters, hence the parameter list is empty. We will give examples of functions with non-empty parameter lists below. The "function code" is all the statements that belong to the function.

Consider the following code that uses a function we named drawPerson( ) :

```
void drawPerson()
{
  line( 40,100,  40,170) ;   // body
  line( 40,170,  20,230) ;   // left leg
  line( 40,170,  60,230) ;   // right leg
  line( 40,130,  20,170) ;   // left arm
  line( 40,130,  60,170) ;   // right arm
  fill(150) ;  // grey for head fill
  ellipse(40,100,20,20) ;    // head
}

void setup()
{
  size(400,400) ;
  drawPerson() ;
}
```
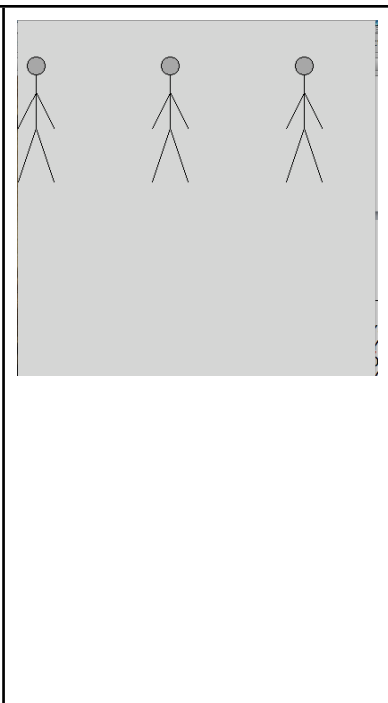
In this example the function drawPerson( ) draw a stick figure person. The parameter list is empty as there are no parameters. The return type is void because there is no value returned. The function is CALLED in the setup( ) function:

    drawPerson( ) ;

Like all Processing sketches the code in setup( ) is run once. Hence, drawPerson( ) is called once. Lets say we want to draw the figure more to the right. We need to modify all the code. It would be much nicer if we could just modify the function so that when called we could specify the (x, y) location where we want the stick figure drawn, something like:

    drawPerson(200, 200) ;

which would then draw the stick figure at location (200,200). To do that we need to modify the function as follows:

2

```
void drawPerson(int inX, int inY)
{
  line( inX+20,inY,  inX+20,inY+70) ;    // body
  line( inX+20,inY+70,  inX,inY+130) ;   // left leg
  line( inX+20,inY+70,  inX+40,inY+130) ;   // right leg
  line( inX+20,inY+30,  inX,inY+70) ;    // left arm
  line( inX+20,inY+30,  inX+40,inY+70) ;    // right arm
  fill(150) ;  // grey for head fill
  ellipse(inX+20,inY,20,20) ;    // head
}

void setup()
{
  size(400,400) ;
  drawPerson(0,50) ;
  drawPerson(150,50) ;
  drawPerson(300,50) ;
}
```

In this example the drawPerson function now has two parameters: inX and inY. Thus, when the function is called, we must supply values for those parameters such as:
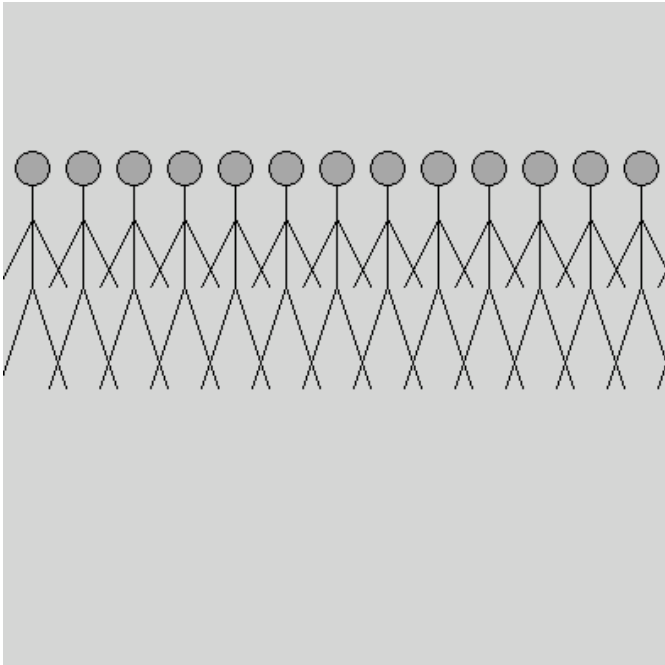
        drawPerson(0, 150) ;

This says call drawPerson, and when the code of drawPerson is run initialize the values of inX and inY with 0 and 150 respectively. In order for this to work we had to change the code inside the drawPerson function to now be dependent on the input parameters. One way to do that for this drawing example is to find the smallest x-coordinate and the smallest y-coordinate, and then change all the code to be relative to the input parameters. For example, in the original function, the statement for the head was:

        ellipse(40,100,20,20) ;

The smallest x value in the original code was 20, and the smallest y was 100, so, to make the statement relative to the input parameters we change it to:

        ellipse(inX+20, inY, 20, 20) ;

Now, if we call the function as drawPerson(20,100,20,20); the person will be drawn in the same place as in the original function. But by using the function with a parameter, we can put in other values when calling the function and hence draw the figure anywhere we want simply by changing the values in the function call rather than changing all the lines of code. We can also put the call inside a for loop as follows:

for (int i = 0 ; i < 400 ; i = i+30) drawPerson(i, 100) ;	

We can also make a the person move using the draw( ) function as follows:

```
int currentX ;   // global var to hold x-coord of figure

void setup()
{
  size(400,400) ;
  currentX = 0 ;
}

void draw()
{
  background(200) ;
  drawPerson(currentX, 100) ;
  currentX = currentX + 2 ;
}

void drawPerson(int inX, int inY)
{
  line( inX+20,inY,  inX+20,inY+70) ;    // body
  line( inX+20,inY+70,  inX,inY+130) ;  // left leg
  line( inX+20,inY+70,  inX+40,inY+130) ;  // right leg
  line( inX+20,inY+30,  inX,inY+70) ;    // left arm
  line( inX+20,inY+30,  inX+40,inY+70) ;    // right arm
  fill(150) ;  // grey for head fill
  ellipse(inX+20,inY,20,20) ;    // head
}
```
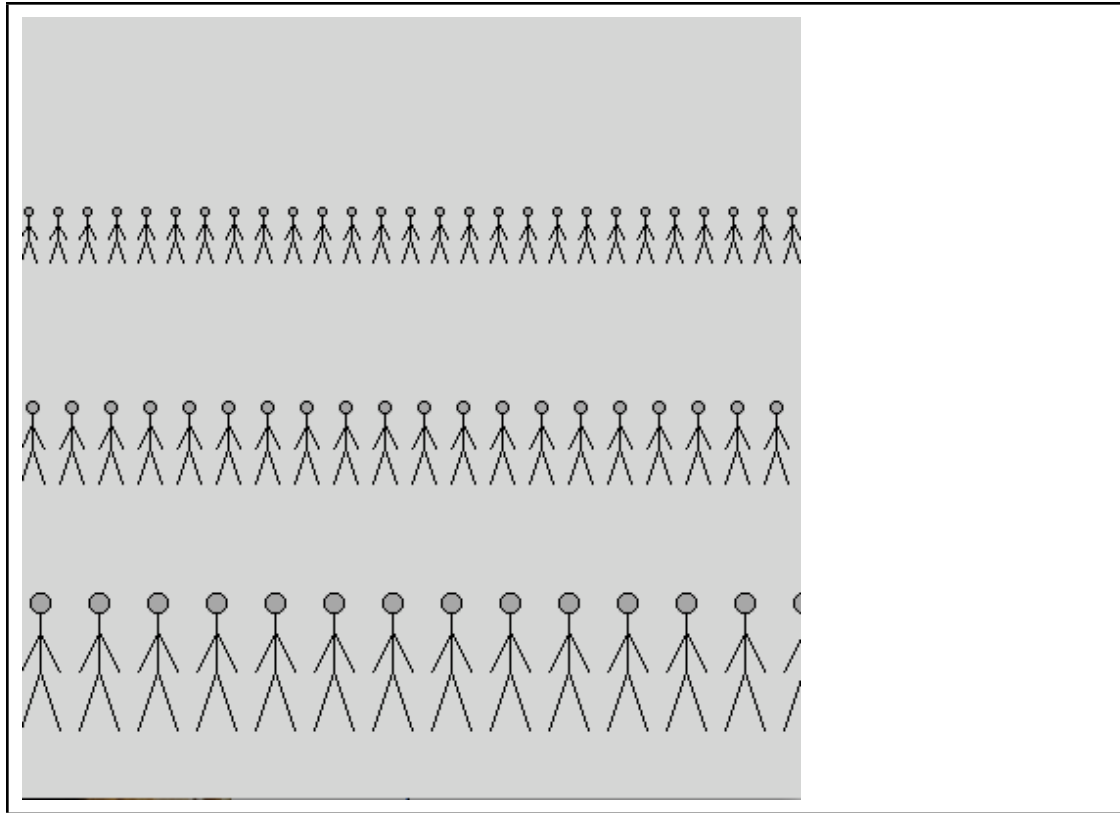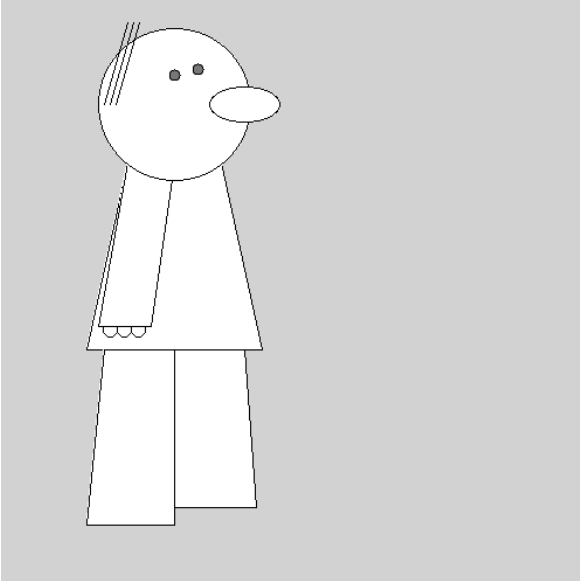
Finally, you may want to be able to specify the size of the figure when you call drawPerson( ). The following code allows you to specify the size. In the example there is a third parameter called "sF", short for scale-factor, added to specify how big the image should be. All numbers that result in the distance of a line are multiplied by "sF", the input value, to make them bigger or smaller. In the example below we make three rows of stick figures, each line above with smaller figures than the line below.

```
void setup()
{
  size(400,400) ;
  for (int i = 0 ; i < 400 ; i = i+15)
    drawPerson(i, 100,0.20) ;
  for (int i = 0 ; i < 400 ; i = i+20)
    drawPerson(i, 200 ,0.30) ;
  for (int i = 0 ; i < 400 ; i = i+30)
    drawPerson(i, 300,0.50) ;

}

void drawPerson(int inX, int inY, float sF)
{
  // sF means "scale factor"
  line( inX+20*sF, inY,  inX+20*sF, inY+70*sF) ;    // body
  line( inX+20*sF, inY+70*sF,  inX, inY+130*sF) ;   // left leg
  line( inX+20*sF, inY+70*sF,  inX+40*sF, inY+130*sF) ;  // right leg
  line( inX+20*sF, inY+30*sF,  inX, inY+70*sF) ;    // left arm
  line( inX+20*sF, inY+30*sF,  inX+40*sF, inY+70*sF) ;   // right arm
  fill(150) ;  // grey for head fill
  ellipse(inX+20*sF, inY, 20*sF, 20*sF) ;   // head
}
```

As another example, here is a way to draw "keyhole ken". Keyhole ken is a simple cartooning exercise for drawing people. The book "The cartoonist's Workbook Drawing, Writing Gags, Selling", by Robin Hall, introduces keyhole ken.

7

```
size(500,500) ;

quad(75,300,225,300,190,140,110,140) ;
// body
ellipse(95,283,12,12) ; //left finger
ellipse(107,283,12,12) ; // midle finger
ellipse(119,283,12,12) ;  // right finger
quad(85,280, 130,280, 150,140, 110,
140) ;  // arm
ellipse(150,90,130,130) ;  // head
ellipse(210,90,60,30) ;  // nose

fill(100) ;  // make fill black for the eyes
ellipse(150,65,9,9) ;  // center eye
ellipse(170,60,9,9) ;   // right eye

// add a few lines for hair
line(90,90, 110,20) ;
line(95,90, 115,20) ;
line(100,90, 120,20) ;

fill(255) ;   // set fill back to white
quad(140,300, 140,435,  220,435,
210,300 ) ; // right leg
quad(90,300, 75,450,  150,450,
150,300 ) ; // left leg
```


This can also be made into a function with parameters for the (x,y) location as follows:

```
void drawKeyholeKen(int x, int y)
{
quad(x,y+280, x+150,y+280, x+115,y+120, x+35,y+120) ;  // body

ellipse(x+20,y+263,12,12) ; //left finger
ellipse(x+32,y+263,12,12) ; // midle finger
ellipse(x+44,y+263,12,12) ;  // right finger
quad(x+10,y+260, x+55,y+260, x+75,y+120, x+35, y+120) ;  // arm
ellipse(x+75,y+70,130,130) ;  // head
ellipse(x+135,y+70,60,30) ;  // nose
fill(100) ;  // make fill black for the eyes
ellipse(x+75,y+45,9,9) ;  // center eye
ellipse(x+95,y+40,9,9) ;   // right eye

// add a few lines for hair
line(x+15,y+70, x+35,y) ;
line(x+20,y+70, x+40,y) ;
line(x+25,y+70, x+45,y) ;

fill(255) ;   // set fill back to white
quad(x+65,y+280, x+65,y+415,  x+145,y+415, x+135,y+280 ) ; // right leg
quad(x+15,y+280, x,y+430,  x+75,y+430, x+75,y+280 ) ; // left leg
}
```

Then, with the following setup() and draw() commands you can make two keyhole kens to cyclicly walk the screen together:

```
int x1 ;   // x of first ken
int x2 ;  // x of second ken

void setup()
{
  size(900,500) ;
  background(100) ;
  x1 = 0 ;
  x2 = 175 ;
}

void draw()
{
  background(100) ;
  drawKeyholeKen(x1,10) ;
  drawKeyholeKen(x2,30) ;
  x1 += 2 ;
  x2 += 2 ;
  if (x1 > width +  150)
    x1 = -150 ;
  if (x2 > width + 150)
    x2 = -150 ;
}
```

EXERCISE 9A
Come up with your own function to draw a simple image. Perhaps a house, or a tree, both are pretty easy to do. Make three versions of your function: 1) hardcoded values, i.e. no parameters; 2) a function that takes parameters inX and inY and draws your figure at location (inX,inY); 3) a function that takes in inX, inY, and a scale factor.

Write code in your setup( ) function that calls your figure drawing function and shows how it works with the three parameters (inX, inY, scaleFactor). |

Chapter 10: Functions Returning a Value

The previous function example all drew stuff on the screen but did not return a value. We saw functions without parameters and with parameters. Lets consider some numerical functions that return values. Lets say I want to write a function that returns the average of three numbers. I could do this as follows:

```
float avg(float in1, float in2, float in3)
{
  float total, averageValue ;
  total = in1 + in2 + in3 ;
  averageValue = total / 3.0 ;
  return( averageValue ) ;
}

void setup()
{
  float theAvg ;

  theAvg = avg( 10.0, 9.0, 6.0) ;
  System.out.println( theAvg) ;
}
```

When the above code is run the value 8.3333 is printed out to the black output bar. What is important to see here is:

* The function does NOT start with void, instead it says "float", this means the function will return a floating point value.
* Inside the function, the last line is " return(averageValue) ; ", this is returning the value of variable averageValue to the calling statement.
* Insider setup(), the function is called in the line "theAvg = avg(10.0, 9.0, 6.0) ;". This statement says call the function avg( ), passing in parameters 10.0, 9.0 and 6.0, and take the value returned by the function call and assign it to the variable "theAvg".

Lets consider another example:

```
float maxValue(float in1, float in2, float in3, float in4)
{
  float maxVal = 0 ;
  if ( (in1 >= in2) && (in1 >= in3) && (in1 >= in4) )
    maxVal = in1 ;
  if ( (in2 >= in1) && (in2 >= in3) && (in2 >= in4) )
    maxVal = in2 ;
  if ( (in3 >= in1) && (in3 >= in2) && (in3 >= in4) )
    maxVal = in3 ;
  if ( (in4 >= in1) && (in4 >= in2) && (in4 >= in3) )
    maxVal = in4 ;

  return( maxVal ) ;

}

void setup()
{
  float theMax ;

  theMax = maxValue( 10.2, 9.0, 6.0, 7.0) ;
  System.out.println( theMax) ;
}
```
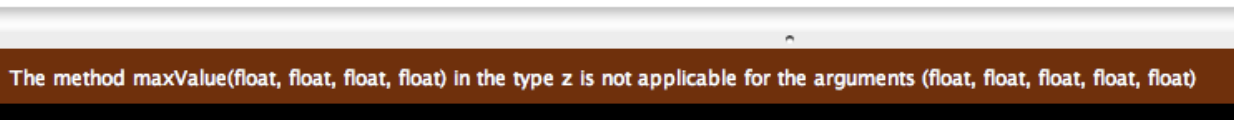
The above will print out "10.2". It is obvious just looking at the code what the answer is, but a computer is not a human, it needs a set procedure to calculate an answer. In this case the if statements find with of the four input parameters is the largest and returns that value.

You may want to print the maximum of 10 numbers, or 20, or 98 numbers. Using the function like this won't work. If I were to say:

theMax = maxValue(10.2, 9.0, 6.0, 7.0, 99.1) ;

I would get the error message:

The method maxValue(float, float, float, float) in the type z is not applicable for the arguments (float, float, float, float, float)

The reason is that the function maxValue is expecting 4 parameter values to be in, no more, not less, and they need to be float values (not a float variable will accept an int as we have stated before). So, how do we deal with this? The answer is to use something called **arrays** which we will do later. For now, just realize that when you declare a

function you specify the number and type of parameters and when you call it your call must match that number and type. The number and type of parameters for a function is called its **signature**.

EXERCISE 10A
Write a function that has three input parameters of type float, that returns the value of the first parameter times the second divided by the third. Say you name the fuction myFunc(), show that this function works correctly by calling and printing out the results using Sytem.out.println() of: myFunc(4, 7, 2)   => should give you 14 myFunc(3, 3, 2)   => should give you 4.5 myFunc(2, 8, 4) => should give you 4

Chapter 11: Using Functions With Colors

Functions can take any defined type (or class which we will explain later) as input and return and defined type or class. Hence, we can use functions to manipulate and return colors also. Consider the following:

```
PImage im1, im2, im3 ;


color swapRedGreenColor(color inColor)
{
  float r,g,b ;
  r = red(inColor) ;
  g = green(inColor) ;
  b = blue(inColor) ;
  color tempColor = color(b,g,r) ;
  return(tempColor) ;
}

void setup()
{
  size(800,800) ;
  background(100) ;
  im1 = loadImage("parkguell.jpg") ;
  im2 = loadImage("parkguell.jpg") ;

  color tempColor, tempColor2 ;
  float r,g,b ;
  for (int i2 = 0 ; i2 < im1.width ; i2++)
   for (int j2 = 0 ; j2 < im1.height ; j2++)
   {
     tempColor = im1.get(i2,j2) ;
     // call swapRedGreenColor to modify the color
     tempColor2 = swapRedGreenColor(tempColor) ;
     // set the i2,j2 pixel of im2 to be the new color
     im2.set(i2,j2,tempColor2) ;
   }
  image(im1,0,0) ;
  image(im2,im1.width,0) ;
}
```

In this example there is a function named "swapRedGreenColor( )". Notice that it take on parameter and that parameter is of type color. Also notice that it returns a variable of type color also! You can see this return in the last line of the function where it says "return(tempColor);", where tempColor is a variable of type color. The function creates a color variable named "tempColor" and set the red value of tempColor to be the green value of the color passed in the parameter "inColor", and sets the green value of tempColor to be the red value of inColor. The setup( ) function just loads the Park Guell image into PImage variables im1 and im2, and then loops through all the pixels of im2

14

swapping the red and green values for each pixel by calling the function swapRedGreenColor() before calling im2.set( ).

The above example showed how to write a function that takes a color as a parameter and returns a color as a parameter, but the setup( ) function is rather complicated. It can be made less complicated by changing the responsibilities of the function so that the function does a bit more of the work for us. For example, we may want to write a function that takes an image as a parameter and returns a modified image. This way using the function is cleaner/simpler. Consider the following:

```
PImage im1, im2, im3, im4 ;

PImage swapBG( PImage inImage)
{
  float r,g,b ;
  color tempColor, tempColor2 ;
  PImage newImage = inImage.get(0,0,inImage.width,inImage.height) ;

  for (int i2 = 0 ; i2 < inImage.width ; i2++)
    for (int j2 = 0 ; j2 < inImage.height ; j2++)
    {
      tempColor = im1.get(i2,j2) ;
      r = red(tempColor) ;
      g = green(tempColor) ;
      b = blue(tempColor) ;
      tempColor2 = color(b,g,r) ;
      newImage.set(i2,j2,tempColor2) ;
    }
    return(newImage) ;
}

PImage darkenImage( PImage inImage, float darkenFactor)
{
  float r,g,b ;
  color tempColor, tempColor2 ;
  PImage newImage = inImage.get(0,0,inImage.width,inImage.height) ;

  for (int i2 = 0 ; i2 < inImage.width ; i2++)
    for (int j2 = 0 ; j2 < inImage.height ; j2++)
    {
      tempColor = im1.get(i2,j2) ;
      r = red(tempColor) ;
      g = green(tempColor) ;
      b = blue(tempColor) ;
      tempColor2 = color(r-darkenFactor, g-darkenFactor, b-darkenFactor)  ;
      newImage.set(i2,j2,tempColor2) ;
    }
    return(newImage) ;
}
```

The first function, swapRB( ), takes an input of an image, loops through all the pixels and swaps the red and blue color values, and then returns the image. The second

function, darkenImage( ), takes two parameters, and image and a float, and returns an image where every pixel has been darkened by the second parameter amount. It does this by looping through the pixels and subtracting the value of the second parameter from each of the RGB values in every pixel. The functions are called as follows:

```
void setup()
{
  size(800,800) ;
  background(100) ;
  im1 = loadImage("parkguell.jpg") ;
  im2 = swapRB(im1) ;
  im3 = darkenImage(im1,50) ;
  image(im1,0,0) ;
  image(im2,im1.width,0) ;
  image(im3,0,im1.height) ;
}
```

By moving the doubly nested for-loops inside the functions the code in setup() is much more clean, and, one could argue, much more logical. The functions swapRB() and darkenImage() do all the work associated with that task and are used as tools to manipulate the images from setup().   A lot of computer programming can be viewed as making (and using) elegant virtual tools. Don't have the right tool for the job: just make a new function (or a **method** as they are called in OO programming coming up soon) to do the job.

Lets say we want to average two images together: i.e. create a new image where each pixel is the average value of the pixel at the location in each image. The following function does exactly this:

17

```
PImage blendImages(PImage inIm1, PImage inIm2)
////////////////////////////////////////
// Input: two images
// Output:  An image that is the blend (average) of the two
//    input images.
//  Algorithm: This code assumes the two images are the same
//    size and loops over the width and heith of the first image, getting
//    the color at each (i,j) location and setting the new image pixel
//    color to be the average of the two
//
////////////////////////////////////////
{
  PImage resultIm ;
  color tcolor1, tcolor2, tcolor3 ;
  float r1, r2, g1, g2, b1, b2 ;

  resultIm = inIm1.get(0,0,inIm1.width,inIm1.height) ;


  for (int i2 = 0 ; i2 < inIm1.width ; i2++)
    for (int j2 = 0 ; j2 < inIm1.height ; j2++)
    {
      tcolor1 = inIm1.get(i2,j2) ;
      r1 = red(tcolor1) ;
      g1 = green(tcolor1) ;
      b1 = blue(tcolor1) ;
      tcolor2 = inIm2.get(i2,j2) ;
      r2 = red(tcolor2) ;
      g2 = green(tcolor2) ;
      b2 = blue(tcolor2) ;
      tcolor3 = color( (r1+r2)/2, (g1+g2)/2, (b1+b2)/2) ;
      resultIm.set(i2,j2,tcolor3) ;
    }
    return(resultIm) ;

}
```

The following code then calls this function, and also the darkenImage() function, to create the images below:

```
void setup()
{
  size(800,800) ;
  background(100) ;

  im1 = loadImage("photoScott.jpg") ;
  im1 = im1.get(140,20,400,400) ;   // select 400x400 pixels

  im2 = loadImage("photoBear.jpg") ;
  im2 = im2.get(170,60,400,400) ;   // select 400x400 to align eyes

  im3 = darkenImage(im1, 50) ;   // darken scott
  im4 = blendImages(im3,im2) ;   // blend scott and the bear

  // show the 4 images
  image(im1,0,0) ;
  image(im2,400,0) ;
  image(im3,0,400) ;
  image(im4,400,400) ;
}
```

Note im1 and im2 (top left and top right) are just a 400x400 subset of the images loaded from the .jpg files. The offsets were chosen to roughly line up the eyes. Image im3, show in the bottom left, is the result of calling darkenImage(im1,50), and image im4, shown in the bottom right, is the result of calling blendImages(im3,im2). Note, the bear face and middle-age man's face are not the same shape (thankfully!) so the resultant images is rather funky. There are algorithms to morph two images together but are more advanced and beyond the scope of this class.

EXERCISE 11A

Write a function named brightenImage( ) that takes two parameters, a PImage and a float named "inBrightenFactor", and returns a PImage where each pixel has each R, G, and B value increased by the value in input parameter "inBrightenFactor". Demonstrate the use of your function by creating three images and displaying them, along with the original, where you use different values for the input value including a negative value for one of the three.

EXERCISE 11B

Write a function that takes an image as an input parameter and returns and image that diagonally mirrors the top right diagonaly down to the bottom left as show in the two images below. Your setup() function call this function and display the original and modified image side-by-side.