Congratulations! You passed!

Next Item

1. 1 / 1 points

You may find it helpful to write a program to answer one or more of the following questions. You can access the DukeLearnToProgram JavaScript environment here: http://www.dukelearntoprogram.com/course1/example/index.php (also linked in the Resources tab)

Consider writing the function **crop(image, width, height)** that has three parameters: **image** is a complete image, and **width** and **height** are numbers. This function creates and returns a new image from the parameter image, with the new image having the width and height of the specified **width** and **height** parameters, thus cropping the picture by ignoring the pixels on the rightside or bottomside that are beyond the specified width or height of the picture.

Here is a possible **crop** function with some missing code.

```
function crop(image, width, height){
    var n = new SimpleImage(width,height);
    for(var p of image.values()){
      var x = p.getX();
      var y = p.getY();
      // MISSING CODE
    }
    return n;
}
```

Which one of the following is the correct missing code for the **crop** function?

```
if (x < width || y < height){
    var np = n.getPixel(x,y);
    np.setRed(p.getRed());
    p.setBlue(p.getBlue());
    p.setGreen(p.getGreen());
}
```

```
if (x < width || y < height){
    var np = n.getPixel(x,y);
    np.setRed(p.getRed());
    np.setBlue(p.getBlue());
    np.setGreen(p.getGreen());
}
```

```
if (x < width && y < height){
    var np = n.getPixel(x,y);
    np.setRed(p.getRed());
    np.setBlue(p.getBlue());
    np.setGreen(p.getGreen());
}
```

Correct
This is the correct answer. Both the x and y values must be in the part of the image that is not cropped.

```
if (x < width && y < height){
    var np = n.getPixel(x,y);
    p.setRed(np.getRed());
    p.setBlue(np.getBlue());
    p.setGreen(np.getGreen());
}
```

2. 1 / 1 points

Consider the function **crop(image, width, height)** that has three parameters: **image** is a complete image, and **width** and **height** are numbers. This function creates and returns a new image from the parameter image, with the new image having the width and height of the specified width and height parameters, thus cropping the picture by ignoring the pixels on the rightside or bottomside that are beyond the specified width or height of the picture. Assume this function works correctly.

We select two images to start with:

```
var start = new SimpleImage("astrachan.jpg");
var hide = new SimpleImage("Message.jpg")
```

What is the remaining JavaScript code to take these two images and crop both of them using the **crop** function to make them the same smaller size?

```
var startWidth = start.getWidth();
var startHeight = start.getHeight();
var hideWidth = hide.getWidth();
var hideHeight = hide.getHeight();

start = crop(start,hideWidth, hideHeight);
hide = crop(hide,startWidth, startHeight);
print(start);
print(hide);
```

```
var startWidth = start.getWidth();
var startHeight = start.getHeight();
hide = crop(hide,startWidth, startHeight);
print(start);
print(hide);
```

```
var cropWidth = start.getWidth();
if (hide.getWidth() < cropWidth) {
  cropWidth = hide.getWidth();
}
var cropHeight = start.getHeight();
if (hide.getHeight() < cropHeight) {
  cropHeight = hide.getHeight();
}
start = crop(start,cropWidth, cropHeight);
hide = crop(hide,cropWidth, cropHeight);
print(start);
print(hide);
```

Correct
This is the correct answer.

```
var cropWidth = start.getWidth();
if (cropWidth < hide.getWidth()) {
  cropWidth = hide.getWidth();
}
var cropHeight = start.getHeight();
if (cropHeight < hide.getHeight()) {
  cropHeight = hide.getHeight();
}
start = crop(start,cropWidth, cropHeight);
hide = crop(hide,cropWidth, cropHeight);
print(start);
print(hide);
```

3. 1 / 1 points

Assume we will hide one image inside another image by hiding information inside half of each pixel. The **chop2Hide** function zeros out the lower half of each pixel's RGB components.

Suppose a particular pixel has a red value of 195 right before **chop2Hide** is called. What is that pixel's red value after **chop2Hide** is called on this image?

192

Correct
This is the correct answer.

Suppose the red value before calling chop2Hide is 195. Convert that number to binary by seeing which combination of 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 forms that number. These are the powers of 2 that form an 8-bit number. That is 128 = 2^7, 64 = 2^6, 32 = 2^5, 16=2^4, 8=2^3, 4=2^2, 2=2^1, 1=2^0. Divide 195 by 128. It goes once so the binary number is 128 + ? The remainder of 195 divided by 128 is 67. 64 goes into 67 once with 3 remaining. Thus 195 is 128 + 64 + 2 + 1 = 1 * 2^7 + 1 * 2^6 + 0*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1 * 2^0. The bolded 0's and 1's make up the 8-bit binary representation of the decimal number 195, which is 11000011. The function **chop2Hide** clears out the lower four bits making them all 0s, or 11000000. The resulting number in decimal is 128 + 64 = 192.

Another way to calculate this is to notice that in our code we cleared out the number by dividing the number by 2^4 or 16, converting the result to an integer using **Math.floor()**, and then multiplying it by 16. If we take the number 195 and divide it by 16 we get 12.1875, convert it to the integer 12, and then multiply 12 by 16 we get 192.

97

195

190

202

4. 1 / 1 points

Assume we will hide one image inside another image by hiding information inside half of each pixel. The **shift** function moves the left half of each pixel's RGB components to the right, replacing the left half with all zeros. Suppose a particular pixel has a red value of 162 right before **shift** is called. What is that pixel's red value after **shift** is called on this image?

0

10

Correct
This is the correct answer. The number 162 is 128 + 32 + 2, which in binary is 10100010. When shifting the 1010 to the right and zeroing out the left digits, the resulting binary number is 00001010 which is 8 + 2 = 10.

5

15

22

5. 1 / 1 points

Assume we will hide one image inside another image by hiding information inside half of each pixel. If the red color of a pixel in image 1 is 212 (written as 8 bits that is 11010100) and the red part of the image 2 we want to hide is 168 (in 8 bits that is 10101000). Then what are the 8 bits for this red color after we have hidden image 2 in image 1 by hiding it in half the pixel data?

11011000

11011010

Correct
This is the correct answer. The 1101 is from the left half of image1 and the 1010 is from the left half of image 2 shifted.

10101101

10100100

10001101

10000100

6. 1 / 1 points

The function **newpv** adds two integer parameters together and returns the result. As a safety check, it should also print a message if the result is too big for an RGB value, that is, bigger than 255. Which one of the following is the correct code for the function *newpv*?

```
function newpv(p,q) {
    var answer = p+q;
    return answer;
    if (p + q > 255)  print("error: answer too big");}
```

```
function newpv(p,q) {
    if (answer > 255)  print("error: answer too big");     var
        answer = p + q;
    return answer;
}
```

```
function newpv(p,q) {
    var answer = p + q;
    return answer;
    if (answer > 255)  print("error: answer too big");
}
```

```
function newpv(p,q) {
    var answer = p + q;
    if (p + q > 255)  print("error: answer too big");
        return answer;
}
```

Correct
This is the correct answer.

7. 1 / 1 points

The function **newpv** adds two integer parameters together and returns the result. Assume that this function works correctly, and in addition it prints an error message if a resulting pixel's red, green or blue value is bigger than 255.

The function **combine** has two image parameters a and b. It returns a new image that is the combination of the two images. That is a pixel in the new image will have a red value that is the sum of the red values of the pixels in the same (x,y) location from images a and b, a green value that is the sum of the two green values, and a blue value that is the sum of the two blue values. The function **combine** assumes the two images it is using to create the new image will not have any sums greater than 255. Consider the **combine** function below that has missing code in the body of the for loop.

```
function combine(p,a,b){
    var n = new SimpleImage(a.getWidth(), a.getHeight());
    for(var pa of a.values()){
      // missing code
    }
    return n;
}
```

Which one of the following has the correct missing code?

```
var x = pa.getX();
var pb = b.getPixel(x,y);
var y = pb.getY();
var np = n.getPixel(x,y);
np.setRed(newpv(pa.getRed(),pb.getRed()));
np.setGreen(newpv(pa.getGreen(),pb.getGreen()));
np.setBlue(newpv(pa.getBlue(),pb.getBlue()));
```

```
var x = pa.getX();
var y = pa.getY();
var pb = b.getPixel(x,y);
var np = n.getPixel(x,y);
pa.setRed(newpv(pa.getRed(),pb.getRed()));
pa.setGreen(newpv(pa.getGreen(),pb.getGreen()));
pa.setBlue(newpv(pa.getBlue(),pb.getBlue()));
```

```
var x = pa.getX();
var y = pa.getY();
var pb = b.getPixel(x,y);
var np = n.getPixel(x,y);
np.setRed(newpv(pa.getRed(),pb.getRed()));
np.setGreen(newpv(pa.getGreen(),pb.getGreen()));
np.setBlue(newpv(pa.getBlue(),pb.getBlue()));
```

Correct
This is the correct answer.

```
var np = n.getPixel(pa.getX(), pa.getY());
np.setRed(newpv(pa.getRed(),pb.getRed()));
np.setGreen(newpv(pa.getGreen(),pb.getGreen()));
np.setBlue(newpv(pa.getBlue(),pb.getBlue()));
```