

✖ Try again once you are ready.

Required to pass: 80% or higher

You can retake this quiz up to 3 times every 8 hours.

Back to Week 4

Retake



1 / 1 points

1. 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.dukeleartoprogram.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.

```
1 function crop(image, width, height){
2   var n = new SimpleImage(width,height);
3   for(var p of image.values()){
4     var x = p.getX();
5     var y = p.getY();
6     // MISSING CODE
7   }
8   return n;
9 }
```

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

☐

```
1 if (x < width && y < height){
2   var np = n.getPixel(x,y);
3   p.setRed(np.getRed());
4   p.setBlue(np.getBlue());
5   p.setGreen(np.getGreen());
6 }
```

☒

```
1 if (x < width && y < height){
2   var np = n.getPixel(x,y);
3   np.setRed(p.getRed());
4   np.setBlue(p.getBlue());
5   np.setGreen(p.getGreen());
6 }
```

Correct

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

☐

```
1 if (x < width || y < height){
2   var np = n.getPixel(x,y);
3   p.setRed(np.getRed());
4   p.setBlue(np.getBlue());
5   p.setGreen(np.getGreen());
6 }
```

☐

```
1 if (x < width || y < height){
2   var np = n.getPixel(x,y);
3   np.setRed(p.getRed());
4   np.setBlue(p.getBlue());
5   np.setGreen(p.getGreen());
6 }
```



1 / 1 points

2. 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:

```
1 var start = new SimpleImage("astrachan.jpg");
2 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?

☐

```
1 var cropWidth = start.getWidth();
2 if (cropWidth < hide.getWidth()) {
3   cropWidth = hide.getWidth();
4 }
5 var cropHeight = start.getHeight();
6 if (cropHeight < hide.getHeight()) {
7   cropHeight = hide.getHeight();
8 }
9 start = crop(start,cropWidth, cropHeight);
10 hide = crop(hide,cropWidth, cropHeight);
11 print(start);
12 print(hide);
```

☐

```
1 var startWidth = start.getWidth();
2 var startHeight = start.getHeight();
3 var hideWidth = hide.getWidth();
4 var hideHeight = hide.getHeight();
5
6 start = crop(start,hideWidth, hideHeight);
7 hide = crop(hide,startWidth, startHeight);
8 print(start);
9 print(hide);
```

☐

```
1 var startWidth = start.getWidth();
2 var startHeight = start.getHeight();
3 hide = crop(hide,startWidth, startHeight);
4 print(start);
5 print(hide);
```

☒

```
1 var cropWidth = start.getWidth();
2 if (hide.getWidth() < cropWidth) {
3   cropWidth = hide.getWidth();
4 }
5 var cropHeight = start.getHeight();
6 if (hide.getHeight() < cropHeight) {
7   cropHeight = hide.getHeight();
8 }
9 start = crop(start,cropWidth, cropHeight);
10 hide = crop(hide,cropWidth, cropHeight);
11 print(start);
12 print(hide);
```

Correct

This is the correct answer.



0 / 1 points

3. 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?

☐ 202

☐ 195

☐ 97

☐ 192

☒ 190

This should not be selected

This is not the correct answer. Note that when 195 is represented in binary, it is $128 + 64 + 2 + 1 = 1 * 2^7 + 1 * 2^6 + 0 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$ which is 11000011. When chopping off the lower half, we would be subtracting 0011.



0 / 1 points

4. 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

☐ 5

☐ 10

☐ 22

☐ 15

This should not be selected

This is not the correct answer. The number should be greater than 0, because 162 is greater than 15. There must have been some 1's in the leftmost digits of the 8 bit binary number for 162. Indeed 162 in binary is 10100010.



0 / 1 points

5. 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 binary 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?

☐ 10101101

☒ 11011000

This should not be selected

This is not the correct answer. It is using the lower bits from the original image you want to hide, which is a huge loss of information and unlikely to look anything like that image.

☐ 10100100

☐ 10001101

☐ 10000100

☐ 11011010



1 / 1 points

6. 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**?

☐

```
1 function newpv(p,q) {
2   var answer = p + q;
3   return answer;
4   if (answer > 255) print("error: answer too big");
5 }
```

☐

```
1 function newpv(p,q) {
2   if (answer > 255) print("error: answer too big");
3   var answer = p + q;
4   return answer;
5 }
```

☒

```
1 function newpv(p,q) {
2   var answer = p + q;
3   if (p + q > 255) print("error: answer too big");
4   return answer;
5 }
```

Correct

This is the correct answer.

☐

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



1 / 1 points

7. 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.

```
1 function combine(a,b){
2   var n = new SimpleImage(a.getWidth(), a.getHeight());
3   for(var pa of a.values()){
4     // missing code
5   }
6   return n;
7 }
```

Which one of the following has the correct missing code?

☐

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

☐

```
1 var x = pa.getX();
2 var pb = b.getPixel(x,y);
3 var y = pb.getY();
4 var np = n.getPixel(x,y);
5 np.setRed(newpv(pa.getRed(),pb.getRed()));
6 np.setGreen(newpv(pa.getGreen(),pb.getGreen()));
7 np.setBlue(newpv(pa.getBlue(),pb.getBlue()));
```

☐

```
1 var x = pa.getX();
2 var y = pa.getY();
3 var pb = b.getPixel(x,y);
4 var np = n.getPixel(x,y);
5 pa.setRed(newpv(pa.getRed(),pb.getRed()));
6 pa.setGreen(newpv(pa.getGreen(),pb.getGreen()));
7 pa.setBlue(newpv(pa.getBlue(),pb.getBlue()));
```

☒

```
1 var x = pa.getX();
2 var y = pa.getY();
3 var pb = b.getPixel(x,y);
4 var np = n.getPixel(x,y);
5 np.setRed(newpv(pa.getRed(),pb.getRed()));
6 np.setGreen(newpv(pa.getGreen(),pb.getGreen()));
7 np.setBlue(newpv(pa.getBlue(),pb.getBlue()));
```

Correct

This is the correct answer.