Homework 2: More Image Processing with Higher-Order Functions

Due Wednesday, February 9, 2022 at 11:59pm

Introduction

Following up to Homework 1, in Homework 2 you will perform *all* processing tasks with higher order functions. As before, we use the types `Pixel` and `Image` to specify our functions:

1. A **Pixel** is a three-element array, where each element is a number in the range 0.0 to 1.0 inclusive.
2. An **Image** is an object whose 2D array of Pixels is accessed via gePixel / setPixel.

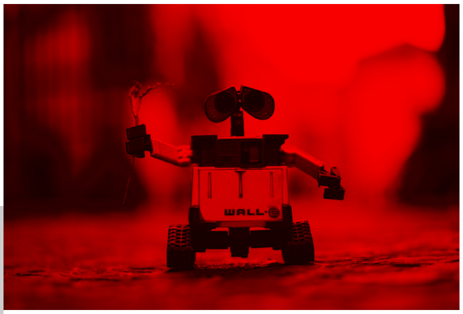
Programming Task

1. Write a function called `imageMapXY` with the following type:

```
imageMapXY(img: Image, func: (img: Image, x: number, y: number) => Pixel): Image
```

The result must be a new image with the same dimensions as `img`. The value of each pixel in the new image should be the result of applying `func` to the corresponding pixel of `img`.

This function is more general than `imageMap`: the new pixel value may also depend on the coordinates of the original pixel. On the right you see an example output of using `imageMapXY` as follows:



```
let url =
'https://people.cs.umass.edu/~joydeepb/robot.jpg';
let robot = lib220.loadImageFromURL(url);
imageMapXY(robot, function(img, x, y) {
  return [img.getPixel(x, y)[0], 0, 0];
}).show();
```

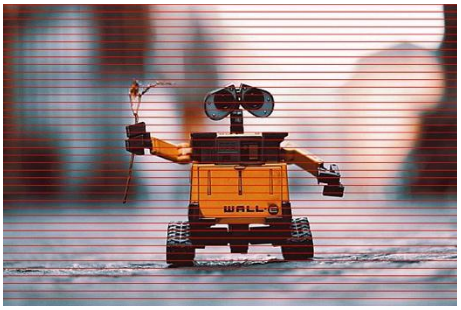
2. Write a function called `imageMask` with the following type:

```
imageMask(img: Image, cond: (img: Image, x: number, y: number) => boolean,
          maskValue: Pixel): Image
```

The result must be a new image, in which the value of pixel at *(x, y)* is either (a) identical to the value of the pixel at *(x, y)* in the original image when `cond(img, x, y)` returns `false` or
(b) the value `maskValue` when `cond(img, x, y)` returns `true`.

You may not use loops in this function.
Instead, use imageMapXY defined above.

On the right you see an example output of using `imageMask`:



```
let url =
'https://people.cs.umass.edu/~joydeepb/robot.jpg';
let robot = lib220.loadImageFromURL(url);
imageMask(robot, function(img,x,y){ return (y % 10 === 0); }, [1, 0, 0]).show();
```

3. Write a function called `imageMapCond` with the following type:

```
      imageMapCond(img: Image, cond: (img: Image, x: number, y: number) => boolean,
         func: (p: Pixel) => Pixel): Image
```

   The result must be a new image, where the value of pixel at *(x, y)* is either (a) identical to the value of the pixel at *(x, y)* in the original image when `cond(img, x, y)` returns `false` or (b) the value `func(p)`, where `p` is the original pixel, when `cond(img, x, y)` returns `true`.
   You may not use loops in this function. Instead, use imageMapXY defined above.

4. Write a function called `isGrayish` with the following type:

```
      isGrayish(p: Pixel): boolean
```

   The result should be true if and only if the difference between the maximum and minimum color channel value is at most 1/3.

For the questions below, you may not use loops within your functions. Instead, use one of the higher-order functions defined above or in Homework 1.

5. Write a function called `makeGrayish` with the following type:

```
      makeGrayish(img: Image): Image
```

   The result must be a new image, where each grayish pixel, as determined by the `isGrayish()` function, is left unchanged. Any other pixel is replaced with a grayscale pixel computed by averaging the three color channels, and setting all three channels in the new pixel to this value.

6. Write a function called `grayHalfImage` with the following type:

```
      grayHalfImage(img: Image): Image
```

   The result must be a new image that is the half-grayed version of the argument, where the top part of the image is grayed out and the bottom part of the image is in color. If the *y*-position is less than half of the image height, then transform this part like with the `makeGrayish` function above.

7. Write a function called `blackenLow` with the following type:

```
      blackenLow(img: Image): Image
```

   The result must be a new image where, for each pixel, any channel value lower than 1/3 is set to 0 for the corresponding pixel in the new image. Other channels for the pixel are not modified.

   Note: your functions must not produce any run-time errors, irrespective of the input image dimensions.

Testing Your Code

As you know, an important part of a project is testing your code thoroughly. In this project, in addition to testing with a variety of input images, you should also use a variety of input *functions* to test your higher-order functions for correctness. To get you started, we have provided a few test cases here. You should define your own additional tests, which will count towards the grade.

- The value returned by imageMapXY should be an image, and must be distinct from the input image.

```
test('imageMapXY function definition is correct', function() {
  function identity(image, x, y) { return image.getPixel(x, y); }
  let inputImage = lib220.createImage(10, 10, [0, 0, 0]);
  let outputImage = imageMapXY(inputImage, identity);
  let p = outputImage.getPixel(0, 0); // output should be an image, getPixel works
  assert(p.every(c => c === 0));      // every pixel channel is 0
  assert(inputImage !== outputImage); // output should be a different image object
});
```

- Test an identity function with imageMapXY. The resulting image should be unchanged. For this test, we will re-use the pixel equality testing helper function from project 1.

```
function pixelEq (p1, p2) {
  const epsilon = 0.002; // increase for repeated storing & rounding
  return [0,1,2].every(i => Math.abs(p1[i] - p2[i]) <= epsilon);
};

test('identity function with imageMapXY', function() {
  let identityFunction = function(image, x, y ) {
    return image.getPixel(x, y);
  };
  let inputImage = lib220.createImage(10, 10, [0.2, 0.2, 0.2]);
  inputImage.setPixel(0, 0, [0.5, 0.5, 0.5]);
  inputImage.setPixel(5, 5, [0.1, 0.2, 0.3]);
  inputImage.setPixel(2, 8, [0.9, 0.7, 0.8]);
  let outputImage = imageMapXY(inputImage, identityFunction);
  assert(pixelEq(outputImage.getPixel(0, 0), [0.5, 0.5, 0.5]));
  assert(pixelEq(outputImage.getPixel(5, 5), [0.1, 0.2, 0.3]));
  assert(pixelEq(outputImage.getPixel(2, 8), [0.9, 0.7, 0.8]));
  assert(pixelEq(outputImage.getPixel(9, 9), [0.2, 0.2, 0.2]));
});
```