Chris Piech, Mehran Sahami and Julie Zelenski Section #4

Code in Place May 7, 2021

Preparing for Section 4

This week students get to write exciting programs that manipulate images in interesting ways. It’s a fun time!

There are three problems included in this week’s section materials, but “Our Section Filter” is the only required problem. We recommend also trying to get through at least the pseudocode and math of “Trim Crop” since this problem focuses on a different type of iteration and gives students practice with pixel coordinate math.

## **Learning Objectives**

Students are learning about images, as well as still getting practice writing helper functions. In section, we want the students to:

1. Understand that images are represented as a collection of pixels, each with an RGB value which determines its colour.
2. Practice using the SimpleImage library.
3. Know the two ways to iterate through images (for pixel in img vs double for-loop) and when to use each.
4. Be able to write code that can manipulate images by getting/setting pixels on an existing image or creating new blank images and setting the pixels on that.

## **What can I assume students already know?**

1. **Parameters and Returns (Last week’s lectures)**

How to define functions which take parameters and give back returns.

1. **Images (Last Friday’s lecture)**

How images are represented and how to load and manipulate them using the SimpleImage library.

## **What to be careful about?**

1. **Pass by reference**

We **haven’t** focused on pass by reference in Python so you should avoid writing functions that take a parameter (such as an image) and modify that parameter in place. For example, in the **Section Filter** problem, we suggest you just write the entire program in main().

1. **What else they don’t know**Students also don’t know about lists, dictionaries, tuples, and returning multiple values.
2. **Integer division**

Since pixel RGB values are ints, make sure to use integer division (//) if you are dividing RGB values. For example, when computing the average of the RGB values, we should integer-divide by 3.

## **Getting ready for section**

Make a video chat link early and remind students when section time will be. ProTip: Send an email reminder about the section to your students 24 hours before your section. Make an announcement and tick the Send Email checkbox!

|  |
| --- |
| Read the Image Reference Guide [handout](https://codeinplace2020.github.io/faqs/imageReference.pdf) to familiarise yourself with the SimpleImage library. |

## **Recommended Section Plan**

### ***Social Time [5 minutes]***

While students are joining the meeting, take a few moments to catch up and see how everyone is doing.

### ***Brief conceptual review [10-15 minutes]***

This week’s section focuses on images. Talk to students about how images are collections of pixels, each with a RGB value that dictates its colour. Remind students of the image coordinate system because they may not be used to this x/y convention.

Show them examples of how to loop through pixels, using either the for each loop:

|  |
| --- |
| **for** px **in** img:  *# do something with px* |

or by doing a double-for loop over image (x,y) positions:

|  |
| --- |
| **for** y **in** range(img.height):  **for** x **in** range(img.width):  px = img.get\_pixel(x, y)  *# do something with px* |

Explain that using a double-for loop is useful when students need access to the pixel’s x and y values to perform manipulations.

It is also worth reiterating that we can set the pixel RGB values in an image to change them. So if we want to darken an entire image:

**for** pixel **in** image:  
 pixel.red = pixel.red // 2  
 pixel.green = pixel.green // 2  
 pixel.blue = pixel.blue // 2

Additionally, this will be the first section in which students are expected to define and use functions with parameters and return values. We recommend spending some time briefly discussing why such functions are helpful, both in the context of image problems and in general.

***Section Filter [15 minutes]***

This problem is asking us to write a program that applies a filter to an image. We’ve deliberately chosen not to decompose the filter into a separate function to avoid passing the image by reference and modifying it by reference.

Computing the pixel average is a good opportunity to decompose into a helper function! You can point out that this average can be used multiple times and is conceptually its own task, which makes it a good candidate for decomposition.

We recommend using a for-each loop for this problem since we don’t care about the x and y values of the pixels that we’re manipulating. It’s a good opportunity to ask your students which type of iteration they think we should use to check for understanding.

The brightness threshold value should ideally be a constant.

***Trim Crop [15 minutes]***

If you don’t have enough time to complete this problem, focus on the pseudocode and pixel math for finding source pixels.

This problem is asking us to write a function that takes in an original image crop size, and returns a new image with that many pixels cropped off each side of the image. Although students have starter code that calls the function, make sure to explain the inputs and output of the helper in order to give them practice with function parameters and return values.

We recommend using a double for loop since the x and y positions of the pixels in the new image are used to determine their source pixels in the original image.

This problem also introduces some common image manipulation patterns that are worth emphasizing:

* When you have to generate an image that is a different size from the original, such as with a border or after trimming, you need to create a new blank image of the right size and fill in the pixels appropriately.
* When using double-for loops to iterate over an image, use .get\_pixel(x, y) in order to access and edit the pixel at a given position.
* Drawing out a simple case for the image is a good way to ensure pixel math goes smoothly. You can try having a side-by-side sketch of the old and new pixelated images to help students reason about what counts as a border pixel and how to find the position of non-border pixels from the original image.

Students often ask whether, in a problem with a source and destination image, they should loop over the original or new image. A good heuristic here is to loop over the smaller image, and then process those pixels, since it allows us to do exactly as much work as we need to, rather than figure out which pixels in the larger image to disregard.

**Something fun**: you can have students experiment with different colors for the border! Ask them which lines of code need to change in order to do this.

***If you have extra time:*** You can try talking about the Add-Border problem or other cool image manipulation tasks you can think of.

## **Teaching Tips**

* Post to your “Ed” group early and often!
* You can pseudocode in whatever way makes sense to you (whiteboarding, typing into a Google doc, etc.) as long as the students are the ones coming up with the steps! One format we recommend is writing your pseudocode as comments in the code.
* Avoid using terms like:
  + “This is simple/easy.”
  + “You should already know how to do this.”
* Let students make mistakes and roll with them! You can turn these bugs into learning moments by going back and saying, “What might have gone wrong?”/  
  “Where do you think the bug might be?”
* Value different solutions - encourage students to think about other ways to solve the problem if you finish one solution early, and discuss the potential pros and cons of each!

## **Code in Place conventions to keep in mind (again, not updated yet)**

* All code must be written inside a **main()** function that gets called under:  
   **if \_\_name\_\_ == “\_\_main\_\_”:**   
  We want to avoid having any code (besides constants and imports) outside functions.
* We don’t explain what the if statement above means until later in the course. If students ask about it, just mention that it’s necessary for the computer to run the program, but they don’t need to worry about it for now!
* In Code in Place, we make a clear distinction about “functionality” and “style.” Style has to do specifically with code readability and reusability. In the class, there are conventions that we emphasize and that we do not want you to deviate from:
  + We follow Python’s [PEP 8](https://www.python.org/dev/peps/pep-0008/) style. This means that variables and function names use “snake case” (all lowercase letters with underscores for spaces). Constants (which will be taught later) will be in all capital letters. **Please do not use camel case (e.g. myVar) or any other type of naming conventions.** If you’re not familiar with Python or PEP 8, take some time to skim the online guidelines.

Your function and variable names should be **descriptive but concise**. Avoid names like **var1**, **var2**, **foo()**, **bar()**, and **function\_that\_has\_very\_long\_name()**