Images & TextGrids

Table of contents

The following problems are for you to gain practice with Image and TextGrid manipulations! Have fun :)

Images (lecture 9)

Here's the Image reference.

- Red noise
- Brightness
- Center square
- Repeat image

`TextGrid` (text-based image alternative)

Here's the TextGrid reference.

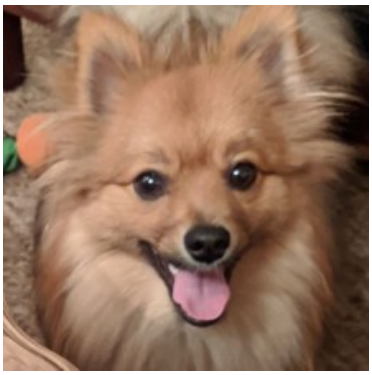
- Keyboard smash
- Digital negative
- Repeat grid
- Straight A's

Images

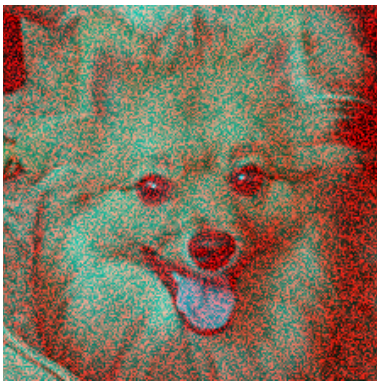
The following problems use the SimpleImage library. Here's the Image reference.

Red noise

Write a program in `red_noise.py` which will edit an image so that each pixel has a random red value. That is, `pixel.red` should take on a random value from 0 to 255, inclusive, for each pixel in the image. If the initial image looks like this (this is the default image):



Your result should look something like this (note since we're using randomness, your result won't look exactly the same, but it should look similar):



Note that the program will show you previews of two images, first the unfiltered image and then the image with red noise applied. You may need to click the right arrow to see the second image with the noise applied.

Once you've done this, if you're feeling extra curious, see what happens if you instead change the green value to be random, or the blue value, or some combination of red, green, and blue! What happens if you randomly set all three of `pixel.red`, `pixel.blue`, and `pixel.green`?

Here's the Image reference.

Brightness

For this problem, you'll implement a program in `brightness.py` which adjusts the brightness of an image by `BRIGHTNESS_TOGGLE`. To do this, you should add `BRIGHTNESS_TOGGLE` to the red, green, and blue values of every pixel in the image, truncate that result using the `truncate()` function we've provided you, which makes sure values stay between 0 and 255, and then set the pixel's red, green, and blue value accordingly. For example, to calculate the new `pixel.red` value, you should pass `pixel.red + BRIGHTNESS_TOGGLE` to `truncate()` and then set `pixel.red` to that result. Do the same thing for blue and green, and do that for every pixel in the image.

For example, if this is the original image (this is the default image):

Here's the result of running this program with `BRIGHTNESS_TOGGLE = 100`:

You can experiment with different values for `BRIGHTNESS_TOGGLE` -- what happens if you change it to a negative value?

We'd like to credit the algorithm for this problem to this source.

Here's the Image reference.

Center square

Write a program in `center_square.py` to copy the center square from an image, given a fixed `SQUARE_WIDTH`. For example, if this is the original image (this is the default image):

The center square with the default `SQUARE_WIDTH` of 200 looks like this:

We've written code which will load the image and create a blank square image, `square_image`, for you to fill in with the right pixels. We *highly recommend* you draw pictures for this problem -- how might you use `image.width` , and `SQUARE_WIDTH` to calculate the coordinate of the center square's top left corner, relative to the original `image`?

Here's the Image reference.

Repeat image

Write a program in `repeat_image.py` that copies an image `NUM_REPEATS` times horizontally. For example, for the default image (images/quad.jpg) and the default value `NUM_REPEATS = 2`, here is the result:

We've written the code to load the image file. Start by initializing a `final_image` SimpleImage with the right width and height (these should be helpful to you: `image.width` and `image.height` can be used to obtain the dimensions of the original image, and `NUM_REPEATS` is the number of times the image is repeated).

Here's the Image reference.

TextGrid

TextGrids are an alternative to SimpleImage which can be printed to the console. The following problems use TextGrids to test your understanding of nested loops and image-like manipulations.

Here's the TextGrid reference.

Keyboard smash

Write a program in `keyboard_smash.py` which generates a TextGrid with 1 row and `NUM_LETTERS` columns of randomly populated letters. To help, we've provided you with a `get_random_value()` function that randomly returns one of the most smashable keyboard characters -- the characters in the middle row of the keyboard.

Here's a sample run of the program (since randomness is involved, your output likely won't be the same):

```
$ python keyboard_smash.py
;SSL;J;JSL
```

Here's the TextGrid reference.

Digital negative

Write a program in `digital_negative.py` which takes a TextGrid of 0s and 1s and changes all 0s to 1s, and all 1s to 0s.

Here's a sample run of the program on `zero_one.txt`, which is the default file:

```
$ python digital_negative.py
Initial grid:
0000
1111
New grid:
1111
0000
```

We've additionally provided you with a `checkerboard.txt` file to test your code on; you can run your program on this file by changing `DEFAULT_FILENAME` to `'checkerboard.txt'`.

Recall that you can access the value of a cell in the TextGrid using `cell.value`, and that all letter values are *strings, not ints!*

Here's the TextGrid reference.

Repeat grid

Write a program in `repeat_grid.py` that copies `grid` onto `final_grid` over and over, `NUM_REPEATS` times, side by side. Here's a sample run of the program on the default grid, `abcd.txt`, with `NUM_REPEATS = 2` (this is the default value):

```
$ python repeat_grid.py
Initial grid:
AB
CD
Final grid:
ABAB
CDCD
```

Here's the TextGrid reference.

Straight A's

Write a program in `straight_As.py` which will scan a `TextGrid` from left to right and top to bottom and, as soon as it encounters the letter 'A' (all the letters in the grid are guaranteed to be uppercase), changes all letters following the first occurrence of 'A' to the letter 'A'.

For example, for an input grid like this (this is the text of guava.txt, the default file):

```
GUAVA
GUAVA
GUAVA
```

The output should be:

```
GUAAA
AAAAA
AAAAA
```

We've provided you three files to test your code on -- guava.txt, antelope.txt, and honolulu.txt. Check them out in the file explorer, and change the value of `DEFAULT_FILENAME` to change which file is read into `grid`!

For this problem, we recommend you initialize a boolean variable and use it to keep track of whether you've seen the letter A already, or not.

Here's the TextGrid reference.