YEAH! HOURS

ASSIGNMENT 3

PART 1: LISTS

SANDCASTLE: GREATER_THAN()

1. "Sandcastle" (warm-up) problem

You should write a function called **`greater_than`** (in the file **`greater_than.py`**) that is passed a threshold integer value and a list of integers, and returns a new list which contains only the numbers strictly greater than the threshold value from the original list passed in. For example, if your function was called as follows:

```
greater_than(6, [20, 6, 12, -3, 14])
```

then it should return the new list:

```
[20, 12, 14]
```

Note that the 6 in the original list is **not** included in the result list since it is not strictly greater than the threshold value, which is 6.

SANDCASTLE: GREATER_THAN()

LISTS REVIEW:

- YOU CAN FIND THE LENGTH OF A LIST USING `len()`
- THERE ARE TWO WAYS TO LOOP OVER A LIST
  - `for i in range(len(some_list)):`
  - `for elem in some_list:`
- YOU CAN CREATE AN EMPTY LIST LIKE THIS: `empty_list = []`
- YOU CAN COPY A LIST LIKE THIS: `list_copy = some_list.copy()`
- YOU ADD ELEMENTS TO A LIST LIKE THIS: `some_list.append(elem)`
- YOU CAN REMOVE THE ELEMENT AT A PARTICULAR INDEX LIKE THIS: `some_list.pop(index)`

SANDCASTLE: GREATER_THAN()

STRATEGIES:

- BUILDING UP
  - CREATE AN EMPTY LIST
  - ADD THE ELEMENTS THAT YOU NEED
  - RETURN THE BUILT-UP LIST

- WHITTLING DOWN
  - CREATE AN COPY OF THE ORIGINAL LIST
  - REMOVE THE ELEMENT THAT YOU DO NOT NEED
  - RETURN THE WHITTLED DOWN LIST

SANDCASTLE: GREATER_THAN()

LOOK OUT FOR:

- LISTS ARE ZERO-INDEXED
- NEVER MODIFY A LIST YOU ARE ITERATING OVER USING A FOR-EACH LOOP

```
for elem in some_list:
    some_list.remove(elem)
```

REMOVE_DUPLICATES()

```
Enter value (0 to stop): 5
Enter value (0 to stop): 3
Enter value (0 to stop): 6
Enter value (0 to stop): 2
Enter value (0 to stop): 7
Enter value (0 to stop): 6
Enter value (0 to stop): 3
Enter value (0 to stop): 3
Enter value (0 to stop): 0
```

If the user entered the values above, the function should return the list:

```
[5, 3, 6, 2, 7, 6, 3, 3]
```

The second function you should write (also in the file **removeduplicates.py**) is called **remove_duplicates(num_list)**. This function is passed a list of integers (**num_list**) and it should create and return a new list which does not include any duplicate values from the original list passed in. The original list passed into the function (**num_list**) should not be changed. For example, calling:

```
remove_duplicates([5, 3, 6, 2, 7, 6, 3, 3])
```

should return the following new list:

```
[5, 3, 6, 2, 7]
```

REMOVE_DUPLICATES()

BASIC STEPS:

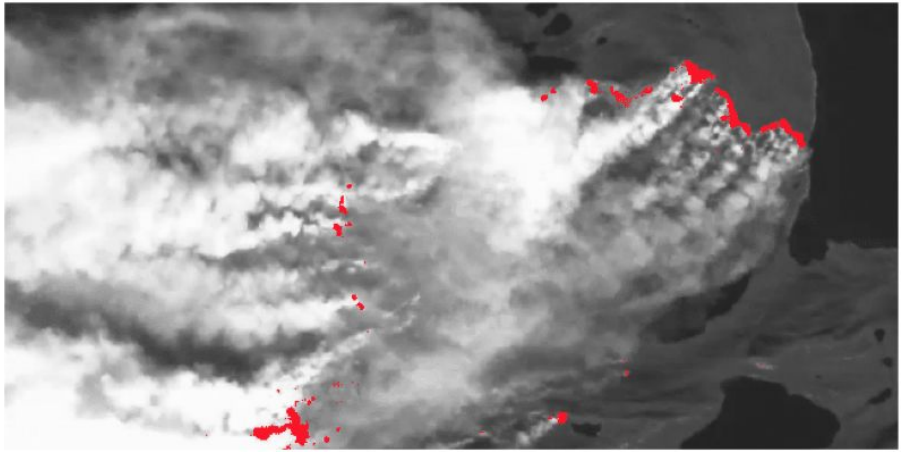
- ACCEPT USER INPUT AND STORE IT IN A LIST
- USE ONE OF THE STRATEGIES USED FOR THE SANDCASTLES TO MODIFY THE LIST TO THE DESIRED STATE
- THE FILTERING CONDITION IS NOT AS SIMPLE AS A GREATER THAN CHECK. MIGHT REQUIRE AN ADDITIONAL LOOP.

REMEMBER TO:

- DECOMPOSE!!! (`read_list()` AND `remove_duplicates()`)
- KEEP TRACK OF FUNCTION PARAMETERS AND RETURN TYPES
- DON'T CHAIN FUNCTIONS TOGETHER
- USE CONSTANTS INSTEAD OF MAGIC VALUES

PART 2: IMAGES

SANDCASTLE: HIGHLIGHT_FIRES()



SANDCASTLE: HIGHLIGHT_FIRES()

IMAGES REVIEW:

- YOU CAN CREATE A NEW BLANK IMAGE LIKE THIS: `img = SimpleImage.blank(width, height)`
- YOU CAN GET THE RGB VALUES OF A PIXEL LIKE THIS: `img.get_pixel(x, y)`
- YOU CAN SET THE RGB VALUES OF A PIXEL LIKE THIS: `img.set_pixel(x, y, pixel)`
- YOU CAN ACCESS AND MODIFY THE INDIVIDUAL RGB VALUES OF A PIXEL LIKE THIS:

```
pixel = img.getpixel(x, y)
red = pixel.red
green = pixel.green
blue = pixel.blue
```

SANDCASTLE: HIGHLIGHT_FIRES()

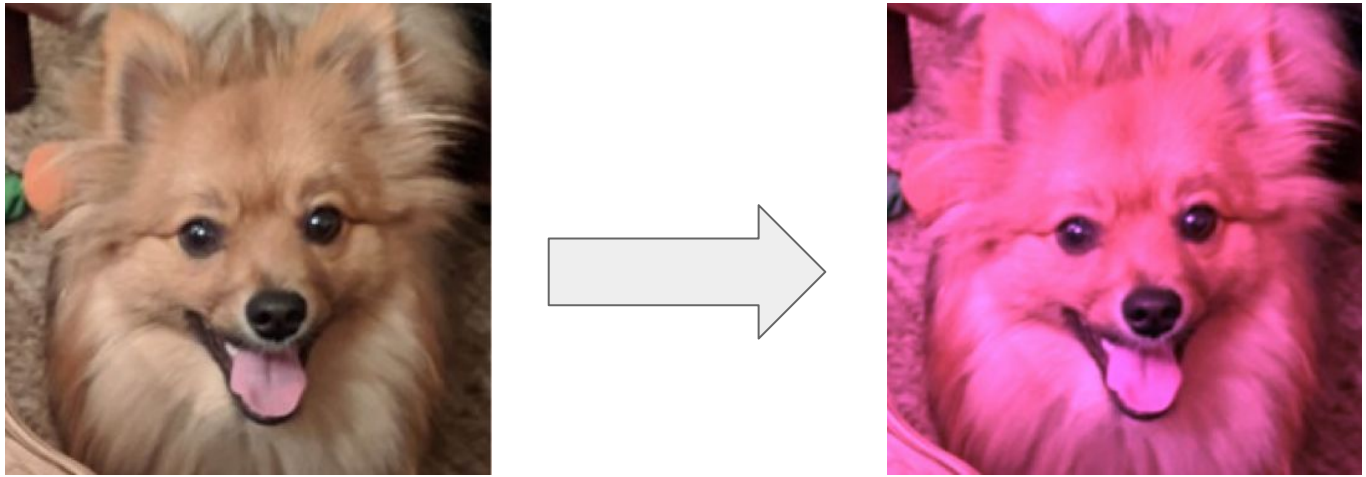
BASIC STEPS:

- CREATE A BLANK IMAGE WHERE WE WILL PUT OUR MODIFIED PIXELS
- ITERATE OVER EVERY PIXEL IN THE IMAGE (YOU'LL NEED 2 FOR LOOPS FOR THIS)
- CHECK IF THE PIXEL MEETS THE CRITERIA TO BE HIGHLIGHTED
- HIGHLIGHT OR GRAY OUT THE CORRESPONDING PIXEL IN THE BLANK IMAGE ACCORDINGLY

REMEMBER TO:

- DOUBLE CHECK YOU'RE USING THE RIGHT COORDINATES. (X CORRESPONDS TO THE WIDTH AND Y CORRESPONDS TO THE HEIGHT)

Warhol images



CREATING A FILTER FOR A SINGLE IMAGE

- VERY SIMILAR TO THE `highlight_fires` SANDCASTLE
- ITERATE OVER EVERY PIXEL AND MULTIPLY THE INDIVIDUAL RGB VALUES BY THE RESPECTIVE SCALE

Warhol images

Warhol images

MAKE WARHOL IMAGE

- CREATE 6 FILTERED IMAGES USING THE FUNCTION YOU JUST WROTE AND STORE THEM IN A LIST
- CREATE A BLANK IMAGE LARGE ENOUGH TO HOLD SIX PATCHES
- COPY PIXELS FROM EACH FILTERED IMAGE (PATCH) INTO THEIR RESPECTIVE POSITIONS ON THE OVERALL IMAGE
- DOING SO WOULD REQUIRE DEVISING A FORMULA THAT TRANSLATES THE COORDINATES OF IMAGE `i` TO COORDINATES ON THE LARGER IMAGE

TIPS:

- ENSURE THAT YOUR BLANK IMAGE HAS THE RIGHT DIMENSIONS
- TRACE BY HAND YOUR MATH FORMULA THAT TRANSLATES PIXEL COORDINATES

GHOST

GHOST

BASIC STEPS:

- CREATE A BLANK IMAGE THAT WILL STORE OUR “GHOSTED” IMAGE
- ITERATE OVER ALL POSSIBLE PIXELS OF THIS IMAGE
- FOR EACH PIXEL COORDINATE, COMPUTE THE AVERAGE PIXEL ACROSS ALL THE PROVIDED IMAGES
- SEARCH ALL IMAGES TO FIND THE PIXEL AT THAT COORDINATE WHICH IS “CLOSEST” TO THE AVERAGE PIXEL OF THAT COORDINATE
- PUT THIS PIXEL IN THE RESPECTIVE LOCATION OF THE BLANK IMAGE TO BUILD UP THE “GHOSTED” IMAGE

GHOST

COMPUTE AVERAGE PIXEL

- **INPUT:** LIST OF PIXELS (EACH PIXEL BELONGS TO A DIFFERENT IMAGE AND REPRESENTS THE SAME X-Y COORDINATE)
- **OUTPUT:** A SINGLE PIXEL WHOSE RGB VALUES ARE THE AVERAGE OF ALL VALUES IN THE RED, GREEN AND BLUE CHANNELS OF THE INPUT IMAGE

- THE COMPUTATION OF THE AVERAGE RGB VALUES IS VERY SIMILAR TO WAY WE CALCULATED THE AVERAGE FOR THE SANDCASTLE
- THE KEY DIFFERENCE IS INSTEAD OF AVERAGING ACROSS THE RGB VALUES OF THE SAME PIXEL, YOU HAVE TO NOW AVERAGE THE RGB VALUES ACROSS MULTIPLE IMAGES FOR A PIXEL AT THE SAME COORDINATES

GHOST

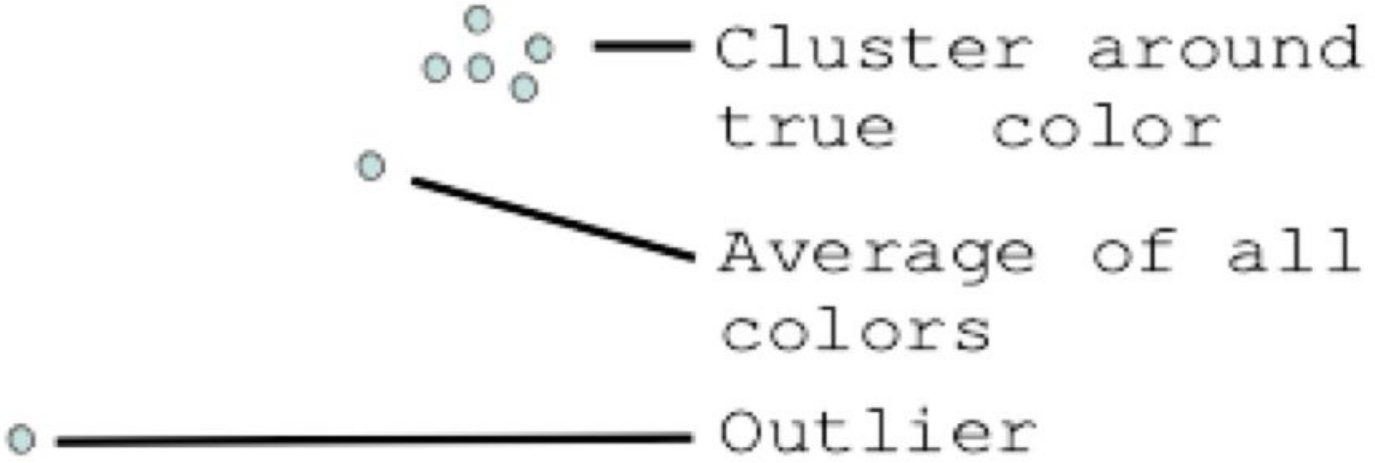
FIND THE BEST PIXEL

- **INPUT:** AVERAGE PIXEL, LIST OF PIXELS (EACH PIXEL BELONGS TO A DIFFERENT IMAGE AND REPRESENTS THE SAME X-Y COORDINATE)
- **OUTPUT:** A SINGLE PIXEL FROM THE INPUT LIST WHICH IS "CLOSEST" TO THE AVERAGE PIXEL PROVIDED

- "CLOSEST" IS THE PIXEL WITH THE LEAST "COLOR DISTANCE" TO THE AVERAGE PIXEL
- YOU PROBABLY WANT TO CREATE A SEPARATE FUNCTION TO CALCULATE THE COLOR DISTANCE BETWEEN TWO PIXELS
- THINK ABOUT HOW YOU CAN USE VARIABLES AND CONTROL FLOW STRUCTURE TO FIND THE *LEAST* VALUE FROM A LIST.

GHOST

$$color\ distance = \sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2}$$



GHOST

CREATE GHOST

- **INPUT:** LIST OF IMAGES
- **OUTPUT:** A SINGLE IMAGE

- ONCE WE HAVE THE OTHER FUNCTIONS, IT IS SIMILAR TO THE SANDCASTLE PROBLEM
- ITERATE OVER EACH PIXEL COORDINATE, APPLY THE FUNCTIONS, OBTAIN A SINGLE PIXEL THAT NEEDS TO BE PUT IN ITS PLACE