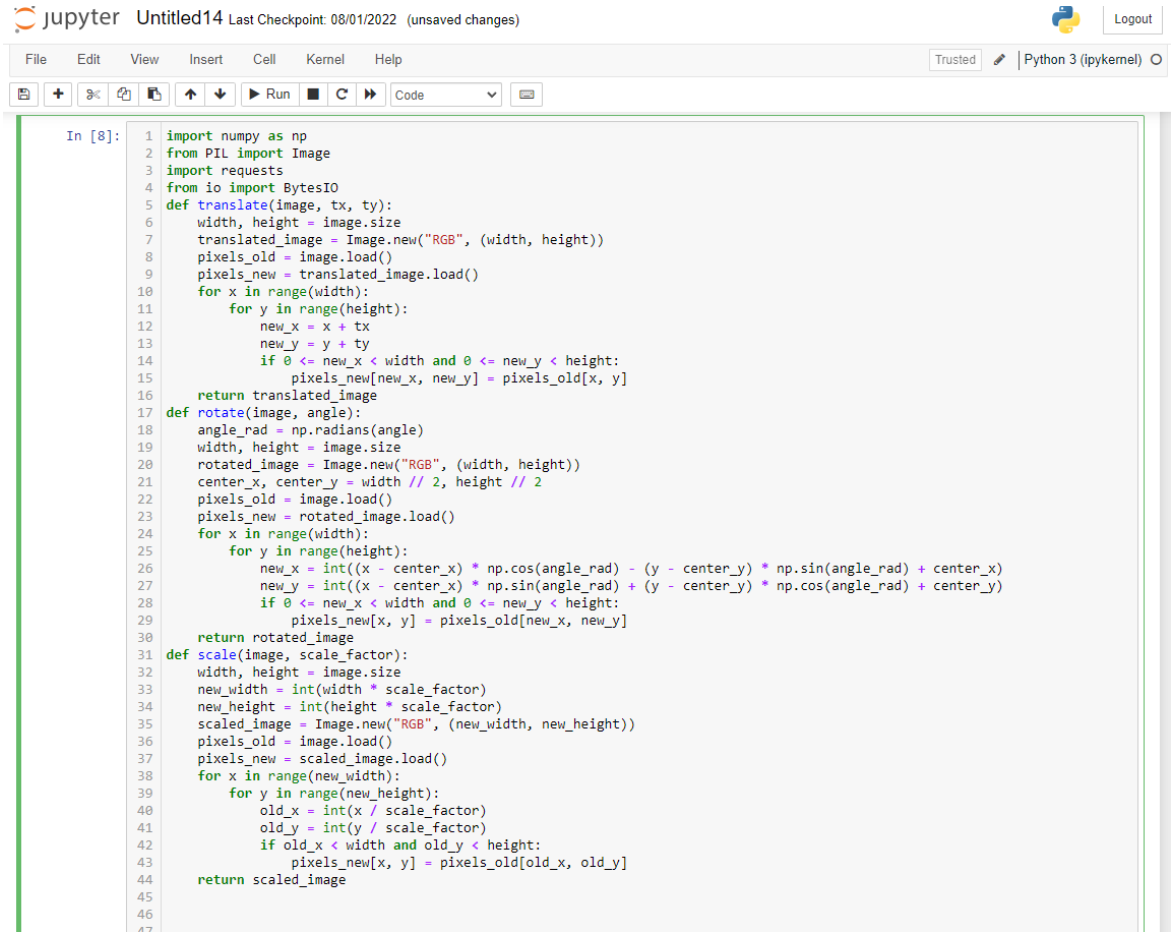
Assignment: Image Formation

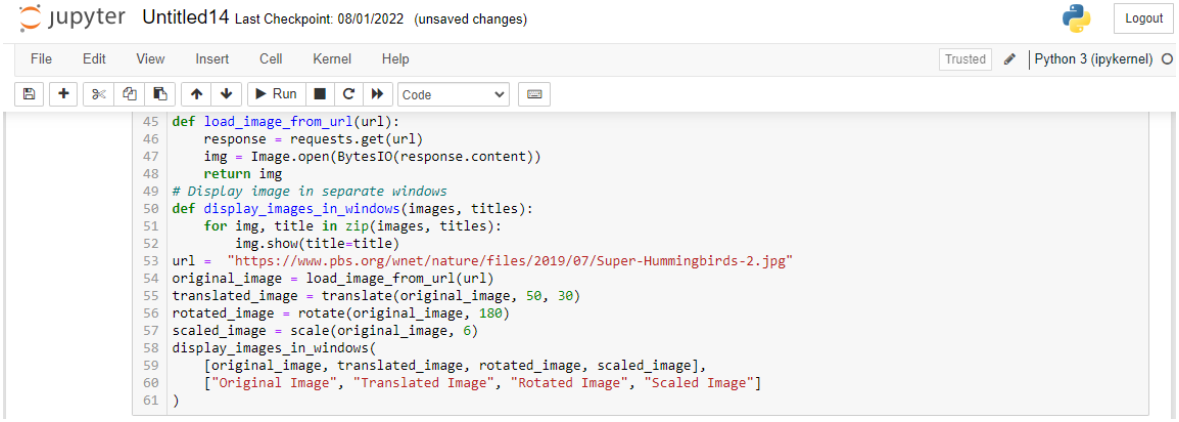
NAME: Sneha Dey ; I'D:221003003010

1: Image Formation

- Objective: Understand how geometric transformations affect images.
- Instructions:
  1. Write a function to perform translation, rotation, and scaling on an image without using any library functions.
  2. Use a sample image and apply each transformation.
  3. Display the original and transformed images side by side.

Code:

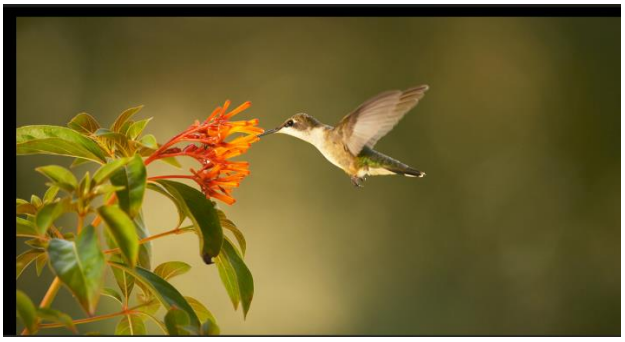




OUTPUT:

Original image:

Transformed images:



2: Photometric Models

- Objective: Understand how lighting conditions affect pixel intensities.
- Instructions:
    1. Write a function to simulate different lighting conditions (e.g., increase brightness, decrease brightness).
    2. Apply these changes to a sample image.
    3. Display the original and altered images.

Code:

jupyter Untitled14 Last Checkpoint: 08/01/2022 (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel)

Run Code

In [23]:

```
from PIL import Image
import numpy as np
import requests
from io import BytesIO
def increase_brightness(image, value):
    img_array = np.array(image)
    bright_img = np.clip(img_array + value, 0, 255).astype(np.uint8)
    return Image.fromarray(bright_img)
def decrease_brightness(image, value):
    img_array = np.array(image)
    dim_img = np.clip(img_array - value, 0, 255).astype(np.uint8)
    return Image.fromarray(dim_img)
def load_image_from_url(url):
    response = requests.get(url)
    img = Image.open(BytesIO(response.content))
    return img
def display_images_side_by_side(images):
    widths, heights = zip(*(i.size for i in images))
    total_width = sum(widths)
    max_height = max(heights)
    combined_image = Image.new('RGB', (total_width, max_height))
    x_offset = 0
    for img in images:
        combined_image.paste(img, (x_offset, 0))
        x_offset += img.size[0]
    combined_image.show()
url="https://img.freepik.com/free-photo/wide-angle-shot-single-tree-growing-clouded-sky-during-sunset-surrounded-by-
    "grass_181624-22807.jpg"
original_image = load_image_from_url(url)
brighter_image = increase_brightness(original_image, 50)
darker_image = decrease_brightness(original_image, 50)
display_images_side_by_side([original_image, brighter_image, darker_image])

```

OUTPUT:

3: Sampling and Quantization

- Objective: Understand the effects of sampling and quantization on image quality.
- Instructions:
    1. Write a function to downsample and upsample an image.
    2. Write a function to quantize an image to different levels (e.g., 2-bit, 4-bit).

    3. Apply these functions to a sample image.
    4. Display the results on image quality.

CODE:

jupyter Untitled14 Last Checkpoint: 08/01/2022 (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel)

Run Code

In [25]:

```
from PIL import Image
import numpy as np
import requests
from io import BytesIO
def downsample(image, factor):
    width, height = image.size
    img_array = np.array(image)
    downsampled_img = img_array[::factor, ::factor]
    return Image.fromarray(downsampled_img)
def upsample(image, factor):
    img_array = np.array(image)
    upsampled_img = np.repeat(np.repeat(img_array, factor, axis=0), factor, axis=1)
    return Image.fromarray(upsampled_img)
def quantize(image, bit_depth):
    img_array = np.array(image)
    levels = 2 ** bit_depth
    scale_factor = 256 // levels
    quantized_img = (img_array // scale_factor) * scale_factor
    return Image.fromarray(quantized_img)
def load_image_from_url(url):
    response = requests.get(url)
    img = Image.open(BytesIO(response.content))
    return img
def display_images_side_by_side(images):
    widths, heights = zip(*(i.size for i in images))
    total_width = sum(widths)
    max_height = max(heights)
    combined_image = Image.new('RGB', (total_width, max_height))
    x_offset = 0
    for img in images:
        combined_image.paste(img, (x_offset, 0))
        x_offset += img.size[0]
    combined_image.show()
url = "https://img.freepik.com/free-photo/wide-angle-shot-single-tree-growing-clouded-sky-during-sunset-surrounded-by-
    "grass_181624-22807.jpg"
original_image = load_image_from_url(url)
downsampled_image = downsample(original_image, 4)
upsampled_image = upsample(downsampled_image, 4)
quantized_image_2bit = quantize(original_image, 2)
quantized_image_4bit = quantize(original_image, 4)
display_images_side_by_side([original_image,downsampled_image,upsampled_image,quantized_image_2bit,quantized_image_4bit])
```

OUTPUT:

4: Image Definition and Neighbourhood Metrics

- Objective: Understand how images are defined and represented, and explore neighborhood metrics.
- Instructions:
    1. Define an image as a 2D matrix and implement functions to compute basic neighborhood metrics (e.g., mean, median, standard deviation within a neighborhood).
    2. Apply these metrics to a sample image.
    3. Display the results and discuss their significance.

CODE:

jupyter Untitled14 Last Checkpoint: 08/01/2022 (autosaved) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel)

Run Code

In [28]:

```
import numpy as np
from PIL import Image
import requests
from io import BytesIO
from scipy.ndimage import generic_filter
def load_image_from_url(url):
    response = requests.get(url)
    img = Image.open(BytesIO(response.content)).convert("L")
    return np.array(img)
def compute_mean(image, size):
    return generic_filter(image, np.mean, size=size)
def compute_median(image, size):
    return generic_filter(image, np.median, size=size)
def compute_std(image, size):
    return generic_filter(image, np.std, size=size)
def display_image(image_array, title="Image"):
    img = Image.fromarray(image_array)
    img.show(title=title)
url = "https://img.freepik.com/free-photo/wide-angle-shot-single-tree-growing-clouded-sky-during-sunset-surrounded-by-
      "grass_181624-22807.jpg"
original_image = load_image_from_url(url)
neighborhood_size = (3, 3)
mean_image = compute_mean(original_image, neighborhood_size)
median_image = compute_median(original_image, neighborhood_size)
std_image = compute_std(original_image, neighborhood_size)
display_image(original_image, title="Original Image")
display_image(mean_image.astype(np.uint8), title="Mean Filtered Image")
display_image(median_image.astype(np.uint8), title="Median Filtered Image")
display_image(std_image.astype(np.uint8), title="Standard Deviation Filtered Image")

```

OUTPUT:

Original Image:

Mean Filtered Image:

Median Filtered Image:

Standard Deviation Filtered Image: