Develop a module to enhance the image by using image arithmetic and logical operations

In [1]:
```
# python3 -m pip install opencv--python
# or
# pip install opencv-python
```

1. ARTHIEMETIC OPERATION:

It involves manipulating pixel values by performing mathematical operations on corresponding pixels of two images. The four arthimetuc operations are:

(a) Addition (b) Subtraction (c) Multiplication (d) Division

Theses opeartions can be used to enhance images, reduce noise and perform other transformation techniques.

(a) Rules of Addition:

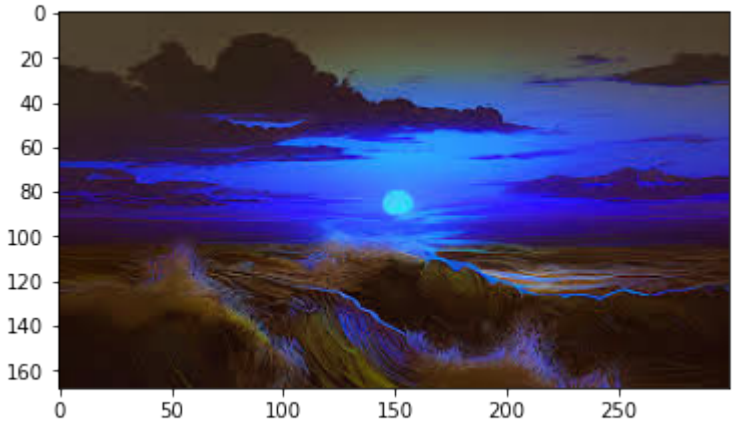
1. If the result of addition is a floating point number, need to be round off.
2. If the result exceeds the pixel range (0-255) the maximum range value is selected.
3. If the result is below the pixel range, minimum range value is selected.

In [2]:
```
# organizing imports
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
# path to input images are specified and
# images are loaded with imread command
im1 = cv2.imread('image1.jpg')
im2 = cv2.imread('image2.jpg')
```

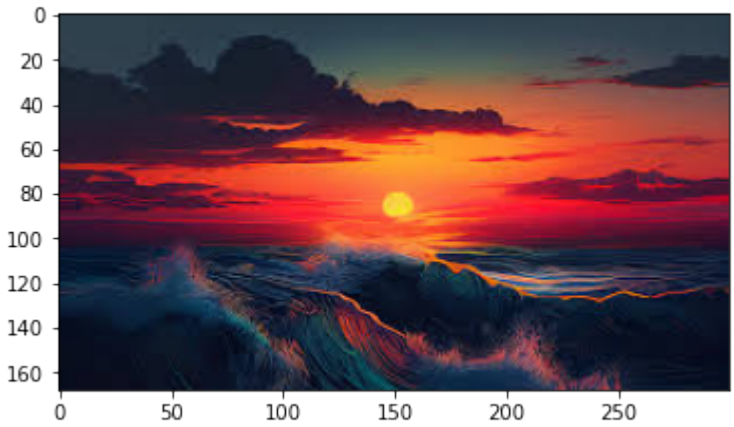
In [4]:
```
plt.imshow(im1)
```

Out[4]: <matplotlib.image.AxesImage at 0x1c3fa939ac0>



In [5]:
```
# converting BGR to RGB
im_rgb1 = cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
plt.imshow(im_rgb1)
```

Out[5]: <matplotlib.image.AxesImage at 0x1c3faa4df70>



In [6]:
```
plt.imshow(im2)
```

Out[6]: <matplotlib.image.AxesImage at 0x1c3faac4eb0>

In [7]:
```
im_rgb2 = cv2.cvtColor(im2, cv2.COLOR_BGR2RGB)
plt.imshow(im_rgb2)
```

Out[7]: <matplotlib.image.AxesImage at 0x1c3fab370a0>

In [8]:
```
# first resize it
im1 = cv2.resize(im1, (300,300))
im2 = cv2.resize(im2, (300,300))
```

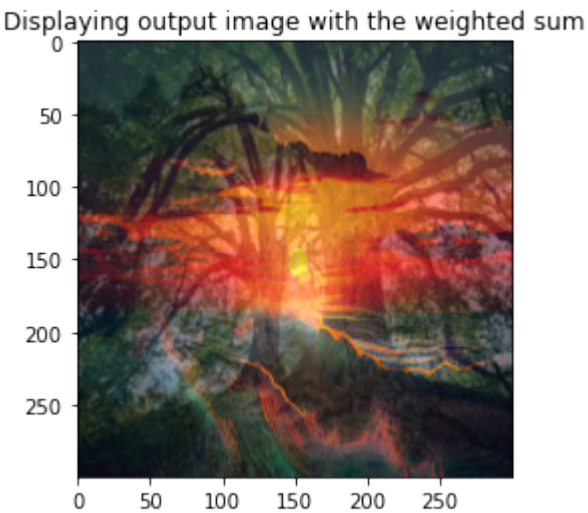
In [9]:
```
# cv2.addWeighted is applied over the
# image inputs with applied parameters

# Blending the images with weights 0.7 and 0.4, and gamma value = 0
# gamma value is responsible for light measurement
weightedSum = cv2.addWeighted(im1, 0.7, im2, 0.4, 0)
```

In [10]:
```
# the window showing output image
# with the weighted sum
im_weight = cv2.cvtColor(weightedSum, cv2.COLOR_BGR2RGB)
plt.title('Displaying output image with the weighted sum')

plt.imshow(im_weight)
```

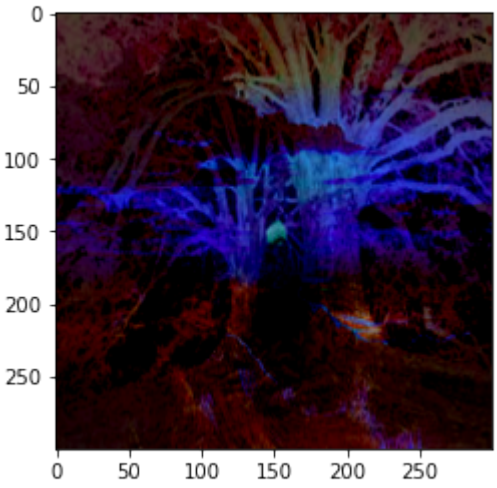
Out[10]: <matplotlib.image.AxesImage at 0x1c3fab5d2b0>



(b) Subtraction: Mainly used for image enhancement, edge detection

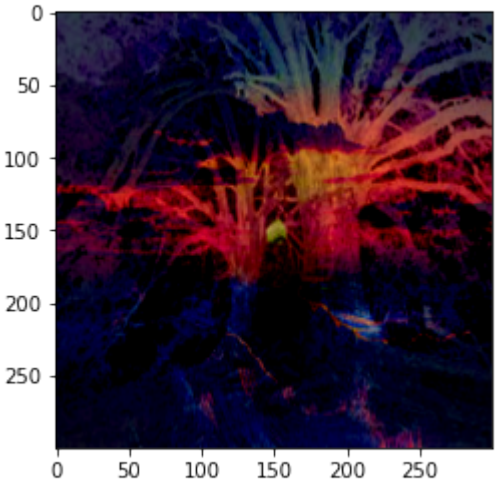
In [11]:
```
# subtract the images
subtracted = cv2.subtract(im1, im2)
plt.imshow(subtracted)
```

Out[11]: <matplotlib.image.AxesImage at 0x1c3fac110d0>



In [12]:
```
im_sub = cv2.cvtColor(subtracted, cv2.COLOR_BGR2RGB)
plt.imshow(im_sub)
```

Out[12]: <matplotlib.image.AxesImage at 0x1c3fac6c910>



2. LOGICAL OPERATION:

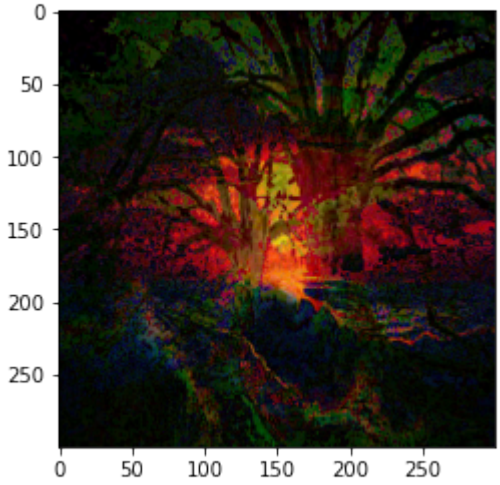
1. AND: Important for extracting common features from multiple images.
2. OR: Important for combining images with different features or detecting specific objects.
3. NOT: For image manipulation and inversion.
4. XOR: For desired feature selection

AND

In [13]:
```
im_and = cv2.bitwise_and(im2, im1, mask = None)
```

In [14]:
```
im_and = cv2.cvtColor(im_and, cv2.COLOR_BGR2RGB)
plt.imshow(im_and)
```

Out[14]: <matplotlib.image.AxesImage at 0x1c3fbca1be0>

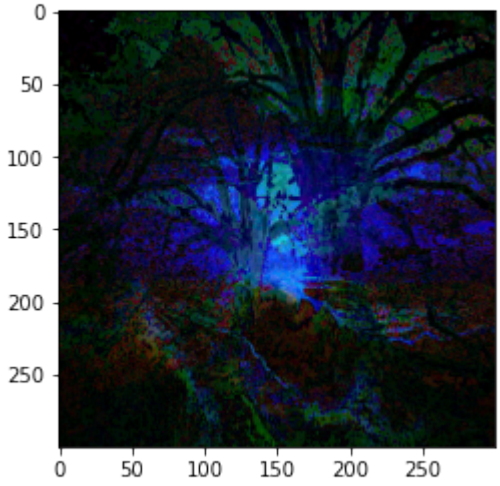


OR

In [15]:
```
im_or = cv2.bitwise_or(im1, im2, mask = None)
```

In [16]:
```
im_or = cv2.cvtColor(im_and, cv2.COLOR_BGR2RGB)
plt.imshow(im_or)
```

Out[16]: <matplotlib.image.AxesImage at 0x1c3fbd0f2e0>

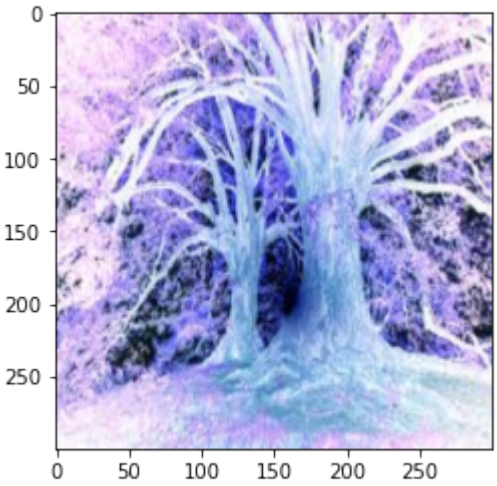


NOT

In [17]:
```
im_not = cv2.bitwise_not(im2, mask = None)
```

In [18]:
```
im_not = cv2.cvtColor(im_not, cv2.COLOR_BGR2RGB)
plt.imshow(im_not)
```

Out[18]:
```
<matplotlib.image.AxesImage at 0x1c3fbd6b9a0>
```



XOR

In [19]:
```
im_xor = cv2.bitwise_xor(im1, im2, mask = None)
```

In [20]:
```
im_xor = cv2.cvtColor(im_xor, cv2.COLOR_BGR2RGB)
plt.imshow(im_xor)
```

Out[20]:
```
<matplotlib.image.AxesImage at 0x1c3fbdcdfd0>
```

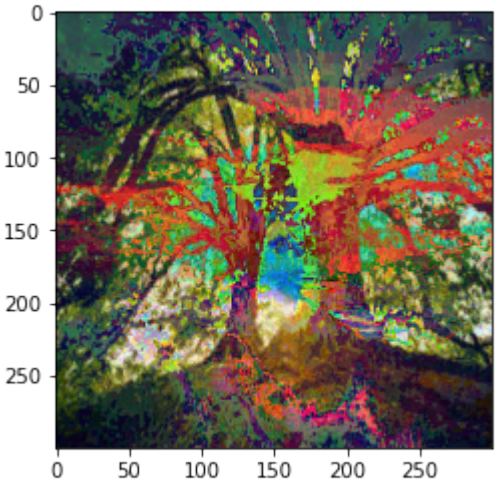


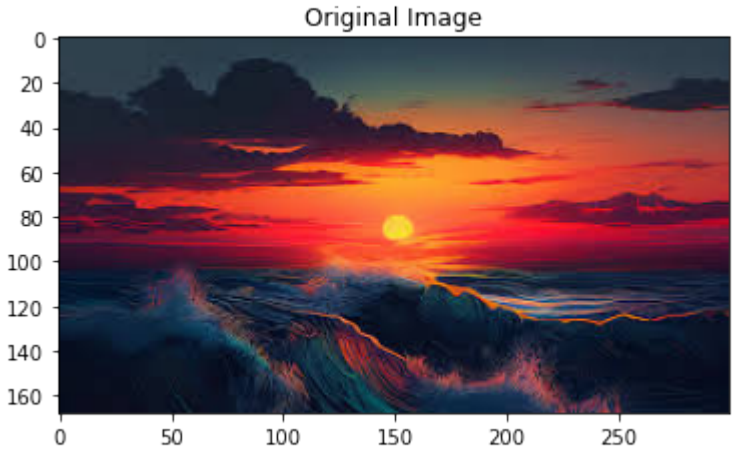
IMAGE ENHANCEMENT USING ARTHIMETIC OPERATION

1. Addition or Brightness

In [21]:
```
img_bgr = cv2.imread("image1.jpg")
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
print("Shape: ", img_rgb.shape)
```

```
Shape:  (168, 300, 3)
```

In [22]:
```
# Display the image
plt.imshow(img_rgb)
plt.title("Original Image")
plt.show()
```



In [23]:
```
matrix = np.ones(img_rgb.shape, dtype = "uint8") * 50

img_rgb_brighter = cv2.add(img_rgb, matrix)
img_rgb_darker = cv2.subtract(img_rgb, matrix)
```

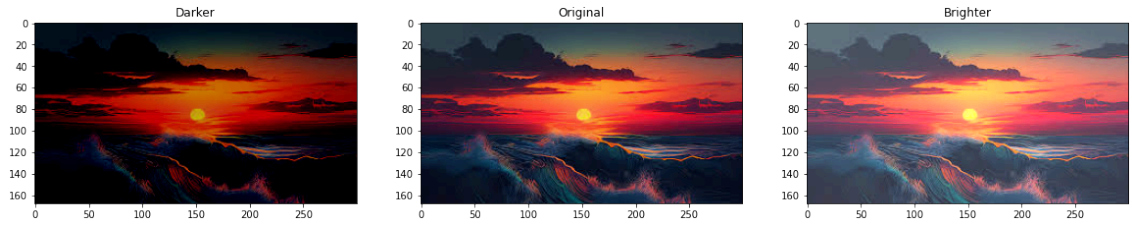
In [24]:
```
plt.figure(figsize = [20, 5])

# Darker Image
plt.subplot(1, 3, 1)
plt.imshow(img_rgb_darker)
plt.title("Darker")

# Original Image
plt.subplot(1, 3, 2)
plt.imshow(img_rgb)
plt.title("Original")

# Brighter Image
plt.subplot(1, 3, 3)
plt.imshow(img_rgb_brighter)
plt.title("Brighter")

plt.show()
```



2. Multiplication or Contrast

In [25]:
```
matrix1 = np.ones(img_rgb.shape) * 0.8
matrix2 = np.ones(img_rgb.shape) * 1.2

img_rgb_lower = np.uint8(cv2.multiply(np.float64(img_rgb), matrix1))
img_rgb_higher = np.uint8(cv2.multiply(np.float64(img_rgb), matrix2))
```

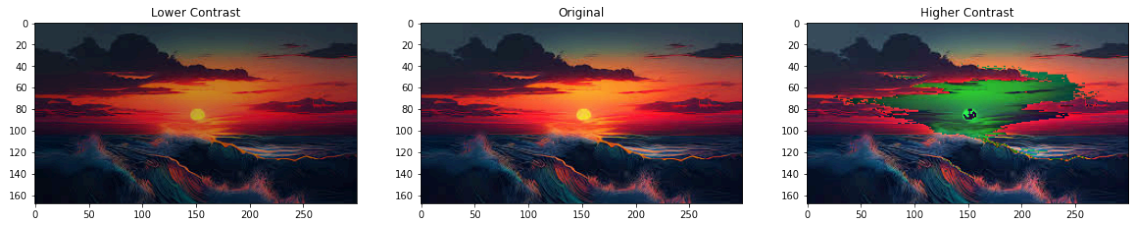
In [26]:
```
plt.figure(figsize = [20, 5])

# Lower Contrast
plt.subplot(1, 3, 1)
plt.imshow(img_rgb_lower)
plt.title("Lower Contrast")

# Original Image
plt.subplot(1, 3, 2)
plt.imshow(img_rgb)
plt.title("Original")

# Higher Contrast
plt.subplot(1, 3, 3)
plt.imshow(img_rgb_higher)
plt.title("Higher Contrast")

plt.show()
```



In [ ]: