Image Steganography

Objective:

To implement an image steganography task using Python

Procedure:

Colab Notebook Link for this lab:
https://drive.google.com/file/d/1LEurmyV-AwRZDSVLv080CVuV7EkVpSEH/view?usp=sharing

Download this image and upload it in your notebook:
https://drive.google.com/file/d/1h4D3Oz0Zb6FpJmJLNPSg8OkkJHF-MJNu/view?usp=sharing

Step-by-step instructions for **hide_message()** function (most of the steps are done for you):

1. The **hide_message()** function takes two arguments: **image_path** (a string representing the file path of the image to be used) and **message** (a string representing the message to be hidden within the image).

2. The function opens the image using the **Image.open()** method from the Pillow library and stores it in the **img** variable.

3. The **message** string is converted to a binary string using the **format()** method. The **ord()** function is used to convert each character in the message to its corresponding ASCII value before converting to binary.

4. The **width** and **height** of the image are obtained using the **size** attribute of the **img** variable.

5. The maximum length of the message that can be hidden in the image is calculated as **max_message_length = (width * height) // 8**, which represents the maximum number of pixels in the image divided by 8 bits per pixel (1 byte per pixel).

6. The function checks if the length of the binary message string is greater than the maximum message length that can be hidden in the image. If it is, a **ValueError** is raised with the message "Message too long to be hidden in the given image".

7. A new image is created using the same mode and size as the original image using the **Image.new()** method from the Pillow library, and stored in the **new_img** variable.

8. The function loops through each pixel in the image using two nested **for** loops.

9. For each pixel, the RGB values are obtained using the **getpixel()** method of the **img** variable and stored in the **pixel** variable as a tuple of three integers representing the red, green, and blue color channels.

10. The function extracts the least significant bit (LSB) of each RGB value using the **format()** method and stores them in **red_binary**, **green_binary,** and **blue_binary** variables, respectively.

11. The function modifies the LSB of each RGB value to hide the binary message. It first modifies the LSB of the red value with the first bit of the binary message. If the end of the binary message has been reached, it sets the LSBs of the green and blue values to '0'. Otherwise, it modifies the LSB of the green value with the next bit of the binary message. If the end of the binary message has been reached, it sets the LSB of the blue value to '0'. Otherwise, it modifies the LSB of the blue value with the next bit of the binary message.

12. The modified RGB values are converted back to integers using the **int()** function with base 2 and stored in the **red**, **green**, and **blue** variables, respectively.

13. A new pixel is created using the modified RGB values and stored in the **new_pixel** variable as a tuple.

14. The new pixel is added to the new image using the **putpixel()** method of the **new_img** variable with the current pixel coordinates (**(x, y)**).

15. After all pixels have been processed, the new image with the hidden message is saved to a file called "hidden.png" using the **save()** method of the **new_img** variable.

16. The function does not return anything.

Step-by-step instructions for **extract_message()** function

1. The **extract_message()** function takes one argument: **image_path** (a string representing the file path of the image containing the hidden message).

2. The function opens the image using the **Image.open()** method from the Pillow library and stores it in the **img** variable.

3. The **width** and **height** of the image are obtained using the **size** attribute of the **img** variable.

4. A variable called **binary_message** is initialized as an empty string to store the binary message extracted from the image.

5. The function loops through each pixel in the image using two nested **for** loops.

6. For each pixel, the RGB values are obtained using the **getpixel()** method of the **img** variable and stored in the **pixel** variable as a tuple of three integers representing the red, green, and blue color channels.

7. The least significant bit (LSB) of each RGB value is extracted using the **format()** method and stored in **red_binary, green_binary,** and **blue_binary** variables, respectively.

8. The LSB of each RGB value is concatenated to the **binary_message** string.

9. The binary message string is then converted back to text using a loop that extracts each 8-bit character from the binary message using slicing and converts it to its corresponding ASCII character using the **chr()** function.

10. The extracted message is stored in the **message** variable.

11. The function returns the **message** variable, which contains the hidden message extracted from the image.

That's it! The **extract_message()** function extracts the hidden message from an image by extracting the LSB of each RGB value in the image and concatenating them into a binary message, which is then converted back to text.
: