UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA - UTEC

Tarea 3

Fecha de inicio: *15 de Febrero, 2022*
Fecha de entrega: *21 de Febrero, 2022*

Curso: *Programación 1 (1111)* – Entrega mediante: *Gradescope*

Indicaciones generales

1. Recuerda que la tarea es **individual**. Los casos de copia/plagio serán sancionados con nota cero (0) en la asignatura.
2. (a) La fecha límite de entrega es el **Lunes 21 de Febrero a las 23:59 hrs**.
   (b) Es altamente recomendable no esperar hasta la última hora.
   (c) *Gradescope* desactivará automáticamente los envíos pasada dicha hora límite.
   (d) **No se aceptarán entregas atrasadas ni entregadas por otros medios.**
3. Revisa bien lo que entregas, aunque en esta oportunidad podrás entregar ilimitadas veces la tarea, la última enviada será la evaluada.
4. Recuerda que *Gradescope* corrige automáticamente tu entrega. Dicha plataforma mostrará si has realizado correctamente las pruebas y mostrará algunos mensajes en color verde. Puedes ver un ejemplo de este caso en el anexo 5.
5. Es posible que hayas subido tu entrega pero hayas modificado algo que no se debió en el template. En ese escenario, Gradescope te mostrará algunos mensajes de error. Puedes ver un ejemplo de esto en el anexo 6.

Gradescope

1. Nosotros les proporcionaremos un código base de donde deberán partir para completar dicho ejercicio. Este archivo es llamado `solution.py` y lo encontrarán en la indicación de la tarea en CANVAS.
2. Al finalizar, **solo** subir el archivo `solution.py` (NO cambiar el nombre del archivo y NO comprimirlo).

3. Cada pregunta tiene diversos casos de prueba. Para obtener la nota completa en una pregunta, el algoritmo debe obtener la respuesta correcta en dichos casos de prueba.

4. Si un caso de prueba falla, visualizarán un mensaje de error con sugerencias. **Lee el error**, revisa el código e inténtalo de nuevo.

5. Los input de los casos de prueba son confidenciales.

Indicaciones específicas

1. En el anexo 1 podrán visualizar la plantilla de código que deberán de seguir para resolver los cuatro ejercicios.

2. Ustedes deben escribir dentro de la sección y a la misma altura de donde esta escrito *"SU SOLUCION EMPIEZA AQUI"*. Además, no deben modificar nada debajo de *"SU SOLUCION TERMINA AQUI"*. Recuerden tener cuidado con las indentaciones.

3. Los input del ejercicio se encuentran en la plantilla. Recuerden usar estas variables para resolver el ejercicio.

4. Ustedes podrán utilizar la imágen llamada 'foto_utec.bmp' para probar su implementación. Es importante que dicha imagen se encuentre en la carpeta donde se encuentra el archivo 'solution.py'. Si desean probar con una imagen externa, verificar que se encuentre en formato BMP de 24 bits.

5. La respuesta de los ejercicios debe ser retornada en una lista de tres dimensiones según se encuentra especificado en la plantilla otorgada.

6. Ustedes podrán utilizar las funciones de apoyo llamadas `leer_imagen` y `guardar_imagen` con la finalidad de convertir una imagen en formato BMP a una lista tridimencional y de una lista tridimencional a una imagen BMP.

7. Si realizan cálculos que conllevan decimales, utilizar la función `round` y convertirlo a entero antes de guardarlo en la lista resultado.

Filtros de Imágenes - (20 pts)

Tu labor en esta ocasión es implementar un programa que aplique filtros a imágenes en formato BMP. Para ello crearemos 1 filtro.

Contexto. Una manera sencilla de representar imágenes es utilizando una cuadrícula de píxeles, los cuales pueden ser de distintos colores. Para una imagen en blanco y negro necesitamos solamente 1 bit, pues el 0 representa el negro y el 1 el blanco.

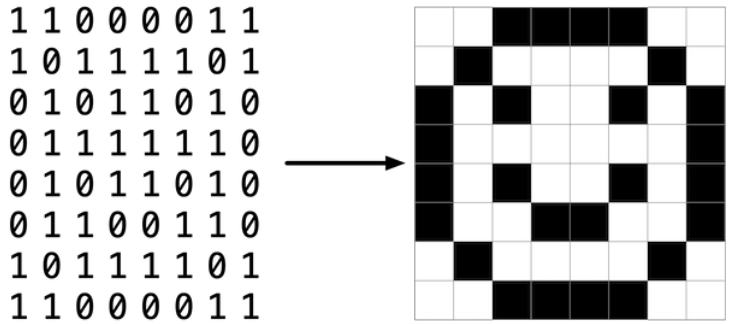

Figure 1: Representación de una imagen en blanco y negro

Ahora pensemos en imágenes más coloridas, para obtener más colores necesitaremos más bits por píxel que los representen. Formatos como BMP, JPEG y PNG soportan colores de *24 bits*. En realidad, BMP soporta colores de 1, 4, 8, 16, 24 y 32 bits. Pero en esta ocasión utilizaremos 24.

Con un BMP de 24 bits representaremos colores RGB: rojo, verde y azul (red, green and blue). Por lo que tenemos 8 bits que representan la cantidad de color de cada uno. 8 rojos $+$ 8 verdes $+$ 8 azules. Veremos estos bits en su forma hexadecimal, como *0xff* y *0x00*.

En la siguiente imagen observamos una imagen de 8x8 bytes (1 byte = 8 bits), donde *ffffff* es el color blanco y *0000ff* es el color rojo.

```
ffffff ffffff 0000ff 0000ff 0000ff 0000ff ffffff ffffff
ffffff 0000ff ffffff ffffff ffffff ffffff 0000ff ffffff
0000ff ffffff 0000ff ffffff ffffff 0000ff ffffff 0000ff
0000ff ffffff ffffff ffffff ffffff ffffff ffffff 0000ff
0000ff ffffff 0000ff ffffff ffffff 0000ff ffffff 0000ff
0000ff ffffff ffffff 0000ff 0000ff ffffff ffffff 0000ff
ffffff 0000ff ffffff ffffff ffffff ffffff 0000ff ffffff
ffffff ffffff 0000ff 0000ff 0000ff 0000ff ffffff ffffff
```

Figure 2: Representación de un bitmap de 24 bits

Filtros. Ahora pensemos en los filtros, estos son modificaciones a las imágenes. Para realizarlos estaremos modificando sus píxeles de tal manera que genere el efecto deseado en la imagen.

Utilizaremos esta imagen como referencia:

Figure 3: Representación de la imagen original

Escala de rojos - 20 pts En este filtro queremos convertir nuestra imagen de colores a una imagen en escala de rojos. Para que un color sea una tonalidad de roja, sus valores en G y B deben ser *0* y en R debe ser distinto de 0. Por lo que, si todos nuestros píxeles cumplen esta condición, nuestra imagen se verá en escala de rojos. Para asegurarnos que el nuevo color en tono rojo tenga las mismas características que nuestra imagen original, colocaremos en R el promedio de sus valores originales. De esta manera nuestra imagen se podrá visualizar como deseamos.

Figure 4: Representación en escala de rojos

1. Anexos

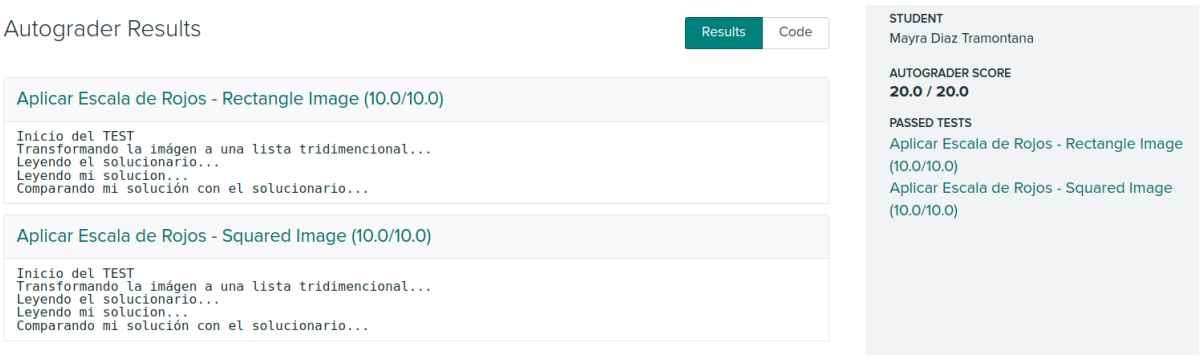

Figure 5: Casos de prueba correctos en Gradescope.

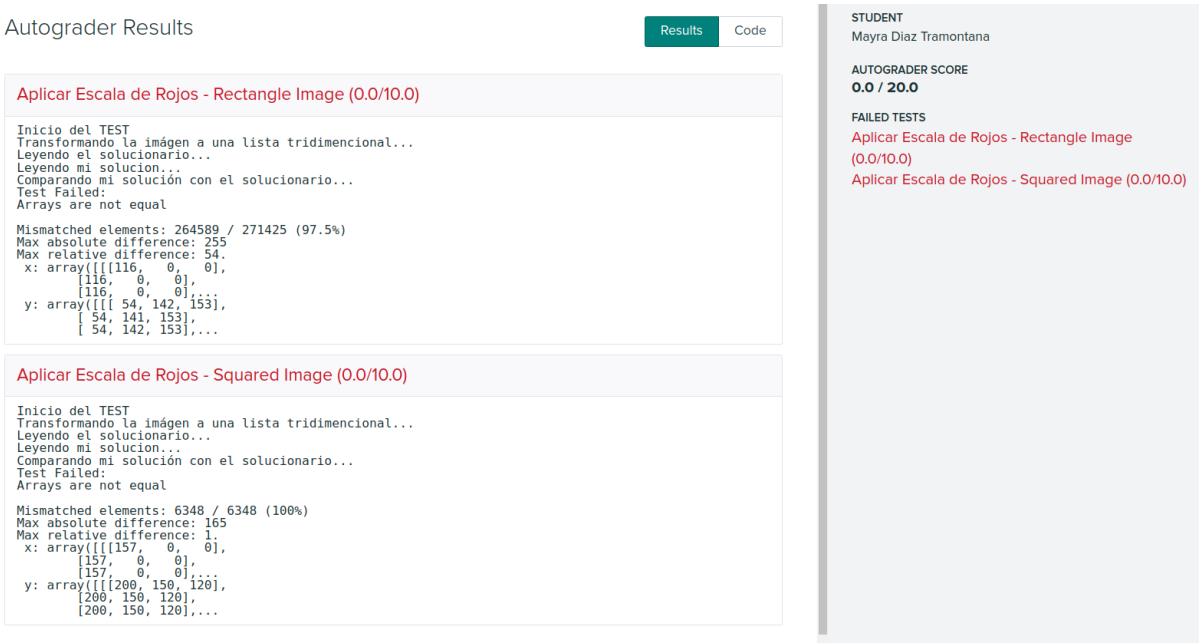

Figure 6: Entrega incorrecta en Gradescope.

```python
import imageio
import numpy as np


def leer_imagen(ruta):
    """
    La función leer_imagen recibe un string con la ruta
    de una imagen en formato BMP y retorna una lista de
    tres dimensiones con el mapa de bits de la imagen.
    Asimismo, convertimos la lista de numpy a una lista
    común y corriente.
    """
    np_array = np.array(imageio.imread(ruta), dtype='int')
    # noinspection PyTypeChecker
    lista_3d = np_array.tolist()
    return lista_3d


def guardar_imagen(ruta, lista_3d):
    """
    La función guardar_imagen recibe una lista de 3
    dimensiones con el mapa de bits de la imagen
    y retorna la imagen en formato bmp.
    """
    return imageio.imwrite(ruta, np.array(lista_3d, dtype='uint8'))


class Solution:

    def aplicar_escala_de_rojos(self, lista_3d=leer_imagen('foto_utec.bmp')):
        result = lista_3d
        # SU SOLUCION EMPIEZA AQUI

        # SU SOLUCION TERMINA AQUI
        return result

guardar_imagen("foto_utec_redscale.png", Solution().aplicar_escala_de_rojos())
```

Listing 1: Template `solution.py`.