Correction

Exercice 1 - Conversion en gris

```python
import numpy as np
from PIL import Image

def naive_grayscale(numpy_image):
    numpy_image = numpy_image.astype(np.int32) # Attention à l'overflow en uint8
    return ((numpy_image[:, :, 0] + numpy_image[:, :, 1] + numpy_image[:, :, 2]) / 3.0).astype(np.uint8)

def ntsc_grayscale(numpy_image):
    return (numpy_image[:, :, 2] * 0.299 + numpy_image[:, :, 1] * 0.587 + numpy_image[:, :, 0] * 0.114).astype(np.uint8)

def red_grayscale(numpy_image):
    return numpy_image[:, :, 2].astype(np.uint8)

def green_grayscale(numpy_image):
    return numpy_image[:, :, 1].astype(np.uint8)

def blue_grayscale(numpy_image):
    return numpy_image[:, :, 0].astype(np.uint8)

# Code pour la visualisation, ne pas modifier ci-dessous
# Lecture de l'image
pil_image = Image.open('starry.jpg')
np_image = np.array(pil_image)

# Conversions en niveaux de gris
naive_gs = naive_grayscale(np_image)
ntsc_gs = ntsc_grayscale(np_image)
red_gs = red_grayscale(np_image)
green_gs = green_grayscale(np_image)
blue_gs = blue_grayscale(np_image)

# Sauvegarde de l'image
first_line = np.hstack([np_image,
                        np.repeat(naive_gs[:, :, np.newaxis], 3, axis=2),
                        np.repeat(ntsc_gs[:, :, np.newaxis], 3, axis=2)]
                       )
second_line = np.hstack([np.repeat(red_gs[:, :, np.newaxis], 3, axis=2),
                         np.repeat(green_gs[:, :, np.newaxis], 3, axis=2),
                         np.repeat(blue_gs[:, :, np.newaxis], 3, axis=2)])
results = np.vstack([first_line, second_line])

pil_image = Image.fromarray(results)
pil_image.show()
```

Exercice 2 - Contraste dans une image

```python
import numpy as np
from PIL import Image
import copy

def grayscale_contrast(numpy_image):
    numpy_image = numpy_image.astype(np.int32)
    min_value = np.min(numpy_image)
    max_value = np.max(numpy_image)
    return (((numpy_image - min_value) / (max_value - min_value)) * 255).astype(np.uint8
    )

def color_contrast(numpy_image):
    numpy_image = numpy_image.astype(np.int32)
    for rgb in [0, 1, 2]:
        min_value = np.min(numpy_image[:, :, rgb])
        max_value = np.max(numpy_image[:, :, rgb])
        numpy_image[:, :, rgb] = (
                    ((numpy_image[:, :, rgb] - min_value) / (max_value - min_value)) *
    255)
    return numpy_image.astype(np.uint8)


pil_grayscale_image = Image.open('bridge.png').convert('L')
pil_color_image = Image.open('tree.jpg')

np_grayscale_image = np.array(pil_grayscale_image)
np_color_image = np.array(pil_color_image)
np_color_image2 = copy.deepcopy(np_color_image)


grayscale_contrasted = grayscale_contrast(np_grayscale_image)
color_contrasted = color_contrast(np_color_image)
color_graycontrasted = grayscale_contrast(np_color_image2)


first_line = np.hstack((np.repeat(np_grayscale_image[:, :, np.newaxis], 3, axis=2),
    np_color_image, np_color_image2))
second_line = np.hstack((np.repeat(grayscale_contrasted[:, :, np.newaxis], 3, axis=2),
    color_contrasted, color_graycontrasted))
results = np.vstack([first_line, second_line])
results = Image.fromarray(results)
results = results.resize((results.width//2, results.height//2))
results.show()
```

Exercice 3 - Warhol

```python
import numpy as np
from PIL import Image

original_image = np.array(Image.open('CAT.PNG'))
hauteur, largeur, _ = original_image.shape
GRID = 3

warhol = np.zeros((hauteur * GRID, largeur * GRID, 3), dtype=np.uint8)

counter = 0
for combination in [
    ([0, 0, 255], 0),
    ([0, 255, 0], 1),
    ([255, 0, 0], 2),
    ([255, 255, 0], 1),
    ([255, 0, 255], 2),
    ([0, 255, 255], 0),
    ([0, 128, 255], 2),
    ([128, 0, 255], 0),
    ([0, 255, 128], 1)]:
        color_background, dimension_maximised= combination

        version_1 = original_image.copy() # Copie profonde nécessaire
        for x in range(hauteur):
            for y in range(largeur):
                r, g, b = version_1[x, y]
                if [r, g, b] == [255, 255, 255]:
                    version_1[x, y] = color_background
                else:
                    version_1[x, y, dimension_maximised] = 255
        row = counter // GRID
        column = counter % GRID
        warhol[hauteur*row:hauteur*(row + 1), largeur*column:largeur*(column + 1)] =
    version_1
        counter += 1

warhol = Image.fromarray(warhol)
warhol.show()
```

Exercice 4 - Chiffrement par One-time pad

```python
import numpy as np
from PIL import Image

# A implémenter
def encode_images(image_1, image_2, one_time_pad):
    image_1_encoded = np.bitwise_xor(one_time_pad, image_1).astype(np.uint8) * 255
    image_2_encoded = np.bitwise_xor(one_time_pad, image_2).astype(np.uint8) * 255
    otp_attack = np.bitwise_xor(image_1_encoded, image_2_encoded).astype(np.uint8) * 255

    return image_1_encoded, image_2_encoded, otp_attack

# A implémenter
def generate_one_time_pad(shape):
    return np.random.randint(0, 256, size=shape, dtype=np.uint8)


# Code pour la visualisation, ne pas modifier ci-dessous
# Lecture des images
epfl = np.array(Image.open('in/EPFL.jpg'))
cervin = np.array(Image.open('in/CERVIN.jpg'))

# Génération des images encodées et attaque
assert epfl.shape == cervin.shape
one_tip_pad = generate_one_time_pad(epfl.shape)
epfl_encoded, cervin_encoded, otp_attack = encode_images(epfl, cervin, one_tip_pad)

# Sauvegarde des images
first_line = np.hstack((epfl, cervin, one_tip_pad))
second_line = np.hstack((epfl_encoded, cervin_encoded, otp_attack))
Image.fromarray(np.vstack([first_line, second_line]).astype(np.uint8)).save('out/ex3_xor
    -otp.png')
```