On the structure of this practise, and on how to submit the solutions

This practise consists on exercises identified by number. You need to upload in canvas **one single `.zip` file** with the filename

`{your-full-name}-graded-practise-1.zip`.

If the following instructions are not properly followed, the automatic grader will give you 0 points

This `zip` file should contain **only** directories named `exercise01`, `exercise02`, ... **Notice that if you compress *the directory containing these directories*, your `.zip` will not follow the correct structure**.

For example, imagine that you have your directories `exercise01`, `exercise02`, ... inside a directory named `pc1`. If you right-click on top of `pc3` and choose *compress*, **the zip file that you will generate will be incorrect and you will receive 0 points**. The reason of this is: your `zip` file will contain, inside it, the directory `pc3` *which will contain, inside it*, the directories `exercise01`, `exercise02`, ... **And this should not be in this way. The directories `exercise01`, `exercise02`, ...should be at the root level inside the `zip` file, without having the *container* directory named `pc1` (or any other name)**.

The way to achieve the correct effect is simple: just select with your mouse *all the directories* `exercise01`, `exercise02`, ... (NOT the directory containing them, but the set all the mentioned directories), press the right button of your mouse and select *compress*. This procedure will generate a `zip` file that will contain the exercises in the root level.

Note: not following the correct directory structure will result in a grade of 0 points.

Convention for the names of the files containing the solutions for the exercises.

Inside the directory `exercise01` there must be a file named `solution.py`. This file should contain the implementation of the function requested in the description of the exercise 1. For the implementation of the problems in this practice is not allowed the usage of any other library besides the opencv and numpy.

How will the evaluation work

Each problem will be automatically judged through the usage of unit tests, written using the framwork PyTest, in Python. Perhaps (not guaranteed, you should not count with it), you will be provided with *some* of these unit tests, that you can use as a *basic* confirmation that your code works. *If* you are provided with these basic unit tests, you should not rely *only* on them to determine the correctness of your code. You are encouraged to confirm the correctness of your algorith using any means that you consider. For example: define your own unit tests using PyTest. The tests that will be provided (if any) will be just very basic cases that intends to help you checking very simple cases.

Exercise 1 - 20 points - directory: `exercise01`

Use the billinear interpolation seen in class to implement, in the file `exercise01/solution.py`, the function:

```
resize(InputImage, NEW_WIDTH, NEW_HEIGHT, PADDING_STRATEGY)
```

Arguments

- `InputImage`: matrix representing the input image
- `NEW_WIDTH`: new width in pixels
- `NEW_HEIGHT`: new height in pixels
- `PADDING_STRATEGY`: can be `'ZEROS'` or `'LAST_PIXEL'`, with the semantics seen in class.

Expected result

A matrix of size `NEW_WIDTH` $\times$ `NEW_HEIGHT`, whose type is exactly the same as `InputImage`. For example: if `InputImage` was RGB, the result should be RGB; If it was 1-channel (grayscale), it should be 1-channel; etc.

Exercise 2 - 20 points - directory: `exercise02`

Implement, in the file `exercise02/solution.py` the function:

```
chessboard_with_custom_colors(
  WIDTH_IN_PIXELS,
  HEIGHT_IN_PIXELS,
  NUM_OF_CELLS_HORIZONTAL,
  NUM_OF_CELLS_VERTICAL,
  COLORS)
```

Arguments

- `WIDTH_IN_PIXELS`: width in pixels of the resulting image
- `HEIGHT_IN_PIXELS`: height in pixels of the resulting image
- `NUM_OF_CELLS_HORIZONTAL`: number of cells that the chessboard should have horizontally
- `NUM_OF_CELLS_VERTICAL`: number of cells that the chessboard should have vertically
- `COLORS`: list of colors.

Expected result

The list of colors is a 0-indexed list of colors. For example:

```
colors = [
  np.array([0, 0, 0]),      # black
  np.array([255, 255, 255]),# white
  np.array([0, 0, 255]),    # red
  np.array([0, 255, 0]),    # green
  np.array([255, 0, 0]),    # blue
  np.array([0, 255, 255]),  # yellow
]
```

The cell i, j of the board must be the i-th row from top to bottom, and the j-th column from left to right (in both cases, starting from 0). The color in which this cell must be painted is:

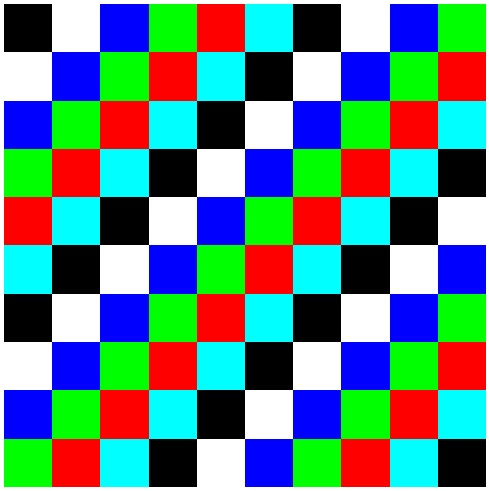
```
colors[ (i + j) % len(colors)]
```

The return value should be a numpy matrix of 3 channels, with `dtype=np.uint8`

This is: you can create the image that you will return using a line similar to

```
result = np.zeros(
  (HEIGHT_THAT_YOU_WANT, WIDTH_THAT_YOU_WANT, 3),
  dtype=np.uint8)
```

One example:



Exercise 3 - 20 points - directory: `exercise03`

Based on the exercise 2 of the class number 5, implement a python script that displays the input image `lowcontrast.png` and, in the same window, one slider that variates the contrast when moving it. Optimize the implementation in order for the change of contrast to happen quick enough such that the moving of the slider is *natural*. There should not be any perceptible delay in updating the contrast when moving the slider.

Expected results

The following command:

```
python exercise03/solution.py
```

should create a window that displays the image mentioned, and a slider. Moving the slider should change the contrast of it.

Exercise 4 - 20 points - directory: `exercise04`

Following the explanation available in the material of the class 6, implement a change in the color scales of the pixels of `lenna.png`, using a mask that consists of white background and a circle centered in the center of the image, tangent to all the four sides of the image (the image is square), filled with color blue. Save the resulting image in `exercise04/output/lenna-colorscale.png`.

Exercise 5 - 20 points - directory: `exercise05`

Based on the exercises 1 and 2 of the class 8, filter the image `lenna.png` with filters box, bartlet and Gaussian of orders 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25.

Do it both with the original image in RGB and in grayscale.

Create a simple HTML page containing a table that shows all the images at the same time. The columns of the table should be: `original`, 3×3, 5×5, 7×7, ..., 25×25. The rows of the table should be *box-grayscale*, *box-rgb*, *bartlett-rgb*, *bartlett-grayscale*, *Gaussian-rgb*, *Gaussian-grayscale*, *laplacian-rgb*, *laplacian-grayscale*.

For the laplacians, fill only the cells 3×3 and 5×5.

All the files (included the HTML page) should be, as explained before, in the directory `excercise08/output`.

Exercise 6 - BONUS EXERCISE - 40 points - directory: `exercise06`

We created during the class 3, in the exercise 2.1, a simple animation in which one circle was moving and bouncing against the borders of the image.

Create an animation with multiple circles that move and can bounce not only against the borders of the image, but also against other circles. Make tham have different speeds and different colors.

Expected results

The command `python exercise06/solution.py` should create a window that displays the animation described.