# Lab 1

- IDE configuration and first OpenCV examples.
- Image operations; reading and displaying images with different formats, direct pixel manipulation.
- Example of a mathematical operation: image subtraction.
- Interaction: selecting pixels and drawing on an image.
- Conversion between color spaces.

**Documentation and Tutorials**

The OpenCV documentation is available at: http://docs.opencv.org

The OpenCV-Python Tutorials are available at:

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html

**Configuration**

Follow the steps in the next links to directly install Python and OpenCV for Python:

https://www.python.org/downloads/

https://pypi.org/project/opencv-python/

An alternative is to install the Miniconda distribution and, afterwards, use the Anaconda Navigator to install the required OpenCV libraries:

https://docs.conda.io/en/latest/miniconda.html

**1.1 First example**

Run and test the file **Aula_01_ex_01.py**

Analyze the code and the OpenCV functions that are used.

Note how an image is read from file and displayed.

Print some of the image attributes: size, number of bytes per pixel, etc.

Modify the code to read a color image from file (e.g., **Orchid.bmp**). Correct any potential errors and print some of the image attributes.

### 1.2 Direct pixel manipulation

Read again the **lena.jpg** image and create a copy of the image (method **copy**).

Access the pixel intensity values of the copy image; set to 0 every pixel of the copy image whose intensity value is less than 128 in the original image.

You can access a given pixel as an element in an array **[i,j]**. Or using the array methods from numpy: **item()** e **itemset()**.

Display the original image and the modified image.

Modify the code to allow reading the name of the gray-level image from the command line.

Do not forget to import the system library (**import sys**) to have access to the command line arguments (**sys.argv[1]**)

### 1.3 Simple mathematical operation: image subtraction

Based on the previous example, create a new program that reads and displays the two image files **deti.bmp** and **deti.jpg**.

To identify possible differences between the two images, carry out a **subtraction** operation.

Be careful, since the **OpenCV subtract** operation is saturated and is different from the **numpy –** operation, that is a modulo operation.

Analyze the resulting image.

**(optional)**

Open an image of your choice in an image editor and save it on file using the **jpeg** format with different compression ratios.

Compare the results of the image subtraction operation for different compression ratios.

### 1.4 Interaction: selecting a pixel and drawing a circle

Modify the previous example to open and display just one image.

Add a **callback function** to detect a **left mouse click** on the window.

When pressing the left mouse button, a filled circle should be drawn, with center on the selected image pixel (function **cv2.circle**).

To define the callback function use:

```
def mouse_handler(event, x, y, flags, params):
    if event == cv2.EVENT_LBUTTONDOWN:
        print("left click")
```

Do not forget to associate the callback function to a window using the following code:

```
cv2.setMouseCallback("Window", mouse_handler)
```

### 1.5 Conversion between color spaces

Load a color image and use the function **cvtColor** to convert it to a gray-level image (COLOR_RGB2GRAY).

**(optional)**

Consult the documentation for the function **cvtColor** and modify the example to visualize the image in different color spaces (for instance: COLOR_RGB2HLS, COLOR_RGB2XYZ, COLOR_RGB2HSV).