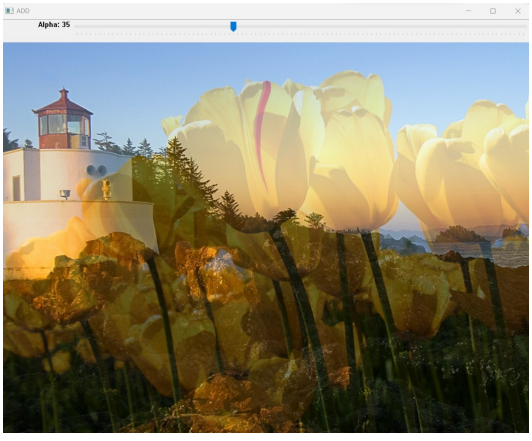
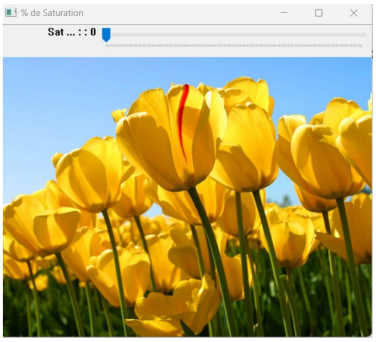
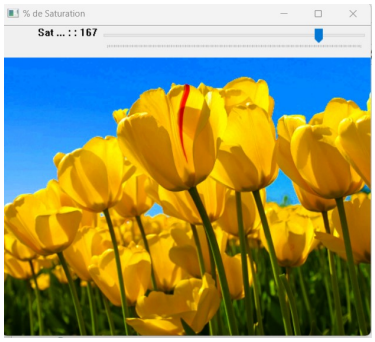
```
# Opérations arithmétiques sur les images : (1-alpha)*A+alpha*B
# où A et B sont des images à couleurs réelles de même taille et alpha prend des valeurs dans [0, 1]

import numpy as np
import cv2

def f(x):
    pass

def main():

    # Lecture et mise à l'échelle
    a = cv2.imread("..\\images\\a.jpg")
    b = cv2.imread("..\\images\\b.jpg")

    # Initialiser l'affichage
    alpha=0;

    c = np.uint8((1-alpha/100.)*a+(alpha/100.)*b)
    cv2.imshow('ADD', c)

    # Faire varier la transparence entre 0 et 100
    cv2.createTrackbar('Alpha : ', 'ADD', 0, 100, f)

    while True:
      # Traitement si la valeur de la transparence change
      if cv2.getTrackbarPos('Alpha : ', 'ADD') != alpha :
          alpha = cv2.getTrackbarPos('Alpha : ', 'ADD')

          c = np.uint8((1-alpha/100.)*a+(alpha/100.)*b)
          cv2.imshow('ADD', c)

      if cv2.waitKey(100) & 0xFF == 27 :
          break

    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```



```
# ------------------------------------------------------------------------------------------------------------------------
# Augmenter la saturation des couleurs par un certain pourcentage

import numpy as np
import cv2

def f(x):
    pass

def saturer(img,p):

    # BGR to HSV
    imgTmp  = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Canal de saturation
    tmp = imgTmp[:,:,1]

    # Augmenter la saturation de p%
    tmp= np.uint16(tmp*(1+p/100.))

    # Mettre les valeurs >255 à 255
    tmpSup255 = tmp>255 # Matrice à valeurs logiques

    tmp = ~tmpSup255*tmp+tmpSup255*255

    # HSV to BGR
    imgTmp[:,:,1] = tmp
    imgTmp = cv2.cvtColor(imgTmp, cv2.COLOR_HSV2BGR)

    return imgTmp

def main():

    # Lecture et mise à l'échelle
    a = cv2.imread("..\\images\\b.jpg")
    width     = int(a.shape[1]/2)
    height    = int(a.shape[0]/2)
    newDim    = (width, height)
    a = cv2.resize(a, newDim)

    # Initialiser l'affichage
    prct=0;
    b = saturer(a,prct);
    cv2.imshow('% de Saturation', b)

    # Faire varier le pourcentage entre 0 et 200
    cv2.createTrackbar('Sat ... : ', '% de Saturation', 0, 200, f)

    while True:
```

```
    # Traitement si la valeur du pourcentage  change
    if cv2.getTrackbarPos('Sat ... : ', '% de Saturation') != prct :
        prct = cv2.getTrackbarPos('Sat ... : ', '% de Saturation')
        b = saturer(a,prct);
        cv2.imshow('% de Saturation', b)

    if cv2.waitKey(100) & 0xFF == 27 :
        break
  cv2.destroyAllWindows()

if __name__ == "__main__":
  main()
```

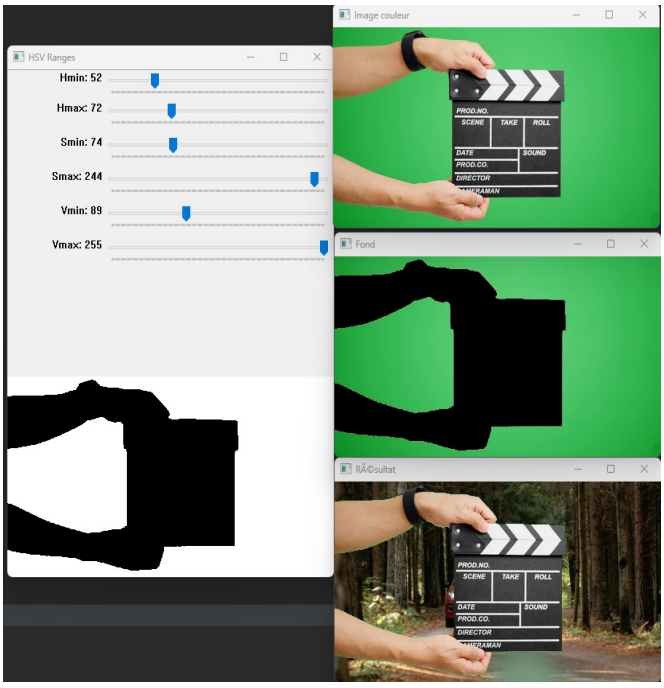




 # ----------

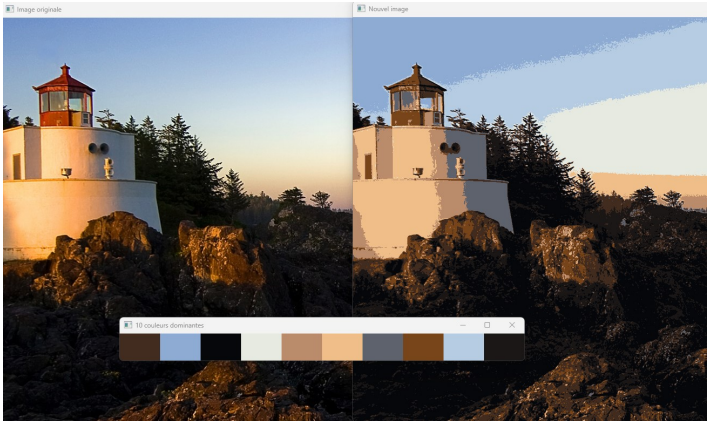
 # A partir d'une image A, à couleurs réelles, calculer une image binaire (masque) qui affiche en blanc les pixels avec
 # une couleur dans une plage définie par des intervalles de l'espace HSV, les autres pixels en noir.
 # Ensuite, la partie de l'image A qui correspond au masque est extraite puis ajoutée à une image B.

```
import cv2
import numpy as np

changesONOFF = False

def f(x):
  global changesONOFF
  changesONOFF = True

def main():
  global changesONOFF

  # Charger l'image
  img = cv2.imread("../images/fondvert.png")
  imgr = cv2.imread("../images/arrierePlan.png")
  cv2.imshow('Image couleur', img)
  cv2.imshow('HSV Ranges', img)

  # Créer les TrackBars
```



```
    cv2.createTrackbar('Hmin', 'HSV Ranges', 0, 255, f),  cv2.createTrackbar('Hmax', 'HSV Ranges', 0, 255, f);
    cv2.createTrackbar('Smin', 'HSV Ranges', 0, 255, f);  cv2.createTrackbar('Smax', 'HSV Ranges', 0, 255, f);
    cv2.createTrackbar('Vmin', 'HSV Ranges', 0, 255, f); cv2.createTrackbar('Vmax', 'HSV Ranges', 0, 255, f)

    # Fixer les valeurs initiales
    cv2.setTrackbarPos('Hmin', 'HSV Ranges', 0);  cv2.setTrackbarPos('Hmax', 'HSV Ranges', 255);
    cv2.setTrackbarPos('Smin', 'HSV Ranges', 0);  cv2.setTrackbarPos('Smax', 'HSV Ranges', 255);
    cv2.setTrackbarPos('Vmin', 'HSV Ranges', 0);  cv2.setTrackbarPos('Vmax', 'HSV Ranges', 255);

    while True:

        if changesONOFF is True:
            imgfond = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

            # Lire les valeurs courantes
            hmin = cv2.getTrackbarPos('Hmin', 'HSV Ranges'); hmax= cv2.getTrackbarPos('Hmax', 'HSV Ranges');
            smin = cv2.getTrackbarPos('Smin', 'HSV Ranges');  smax= cv2.getTrackbarPos('Smax', 'HSV Ranges');
           vmin = cv2.getTrackbarPos('Vmin', 'HSV Ranges');  vmax = cv2.getTrackbarPos('Vmax', 'HSV Ranges');

            # Définir les plages de valeurs
            HSVmin = np.array([hmin, smin, vmin]);      HSVmax = np.array([hmax, smax, vmax]);

            mask = cv2.inRange(imgfond, HSVmin, HSVmax)
            imgfond = cv2.bitwise_and(img, img, mask=mask)

            # Affichage
            cv2.imshow("HSV Ranges", mask)
            cv2.imshow('Fond', imgfond)

            changesONOFF = False

        k = cv2.waitKey(1) & 0xFF
        if k == 27:
            break

    imgr = cv2.bitwise_and(imgr, imgr, mask=mask)
    inverse_mask = cv2.bitwise_not(mask)
    img = cv2.bitwise_and(img, img, mask=inverse_mask)

    imgr = cv2.add(imgr, img)

    cv2.imshow('Résultat', imgr)

    cv2.waitKey(0)

cv2.destroyAllWindows()
if __name__ == "__main__":
    main()
```

```
# ----------------------------------------------------------------------------------------------------------------------------------
        # Calculer et afficher les couleurs dominantes d'une image à couleurs réelles ( Calcul dans l'espace Lab)
import cv2
from sklearn.cluster import KMeans
import numpy as np
def main():
    img = cv2.imread("..\\images\\a.jpg",1)
    cv2.imshow("Image originale", img)

    # Convertir l'image en espace de couleur Lab
    imgLab = cv2.cvtColor(img, cv2.COLOR_BGR2Lab)

    # Nombre de couleurs
    nbreDominantColors = 10

    # Créer une image temopraire
    barColorW=75
    barColorH=50
    imgColors = np.zeros((barColorH, barColorW*nbreDominantColors, 3), dtype=np.uint8)

    # Changement d'echelle, pour avoir peu d'exemples
    height, width, _ = imgLab.shape
    dim    = (int(width/5), int(height/5))

    imgExemples = cv2.resize(imgLab, dim)

    # Exemples d'apprentissage, un  par ligne
    examples = imgExemples.reshape((imgExemples.shape[0] * imgExemples.shape[1], 3))

    # Groupement par la technique des KMEANS
    kmeans = KMeans(n_clusters = nbreDominantColors,  n_init=10)
    kmeans.fit(examples)

    # Les Centres des groupement représentent les couleurs dominantes (B, G, R)
    colors = kmeans.cluster_centers_.astype(int)

    # Créer une image
    new_image = np.zeros((height, width, 3), dtype=np.uint8)
```



```
    # Pour chaque pixel, trouver la couleur dominante la plus proche (Dans Lab)
    for i in range(height):
        for j in range(width):

            labColor = imgLab[i, j]

            # Calculer la distance euclidienne à chaque couleur dominante
            distances = np.linalg.norm(colors - labColor, axis=1)

            # Trouver l'indice de la couleur dominante la plus proche
            closest_color_index = np.argmin(distances)
            labColor = colors[closest_color_index]
            new_image[i, j] = labColor

    new_image = cv2.cvtColor(new_image, cv2.COLOR_Lab2BGR)
    cv2.imshow("Nouvel image", new_image)

    # Affichage des couleurs dominantes
    colors = colors.astype(np.uint8)
    for i in range(0,nbreDominantColors):
        # Conversion de Lab à BGR
        labColor = colors[i].reshape(1, 1, 3)
        bgrColor = cv2.cvtColor(labColor, cv2.COLOR_Lab2BGR)
        bgrColor = bgrColor.reshape(3,)

        cv2.rectangle(imgColors, (i*barColorW,0), ((i+1)*barColorW,barColorH), [int(x) for x in bgrColor],-1)

    str_=" "+str(nbreDominantColors)+" couleurs dominantes";
    cv2.imshow(str_, imgColors)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
if __name__ == "__main__":
    main()
```