Importo la funziona Image dal modulo IPython.display per mostrare sotto IPython file .png e, dopo aver importato il modulo png, definisco le funzioni save e load per salvare su disco e per leggere da disco file .png

In [1]:

```
from IPython.display import Image

import png

def save(img, filename):
    pngimg = png.from_array(img,'RGB')
    pngimg.save(filename)

def load(filename):
    with open(filename, mode='rb') as f:
        reader = png.Reader(file=f)
        w, h, png_img, _ = reader.asRGB8()
        img = []
        for line in png_img:
            l = []
            for i in range(0, len(line), 3):
                l+=[(line[i], line[i+1], line[i+2])]
            img+=[l]
        return img
```

funzioni di servizio: crea(w, h, c = (0,0,0)): crea una lista di h liste ciascuna di w pixel ciascuno di colore c.

In [2]:

```
def crea(w,h,color=(0,0,0)):
    img = []
    for _ in range(h):
        riga = []
        for _ in range(w):
            riga+=[color]
        img+=[riga]
    return img
```

ESERCIZIO: Disegno una funzione inverti(img) che presa un'immagine ne crea il "negativo".

NOTA: Il negativo di un colore si ottine prendendo per ognuno dei tre canali il valore 255 meno il valore originario. Ad esempio il bianco (255,255,255) diventa nero (0, 0, 0).

In [3]:

```
def inverti(img):
    w,h=len(img[0]),len(img)
    img1=crea(w,h)
    for x in range(w):
        for y in range(h):
            r,g,b=img[y][x]
            img1[y][x]=(255-r,255-g,255-b)
    return img1
```

In [4]:

```
Image('foto.png')
```

Out[4]:

In [5]:

```
img= load('foto.png')
img1= inverti(img)
save(img1, 'fotoComp.png')
Image('fotoComp.png')
```

Out[5]:

Molti altri esempi di manipolazione del colore richiedono l'applicazione ad ogni pixel di una trasformazione dei canali. Per sfruttare questa similarità possiamo introdurre una funzione filtro() che altera i colori applicando una funzione func() presa in imput.

ESERCIZIO: progettare la funzione filtro(img, func) che ritorna un'immagine che è l'immagine img con i colori filtrati dalla funzione func()

In [6]:

```
def filtro(img, func):
    w,h=len(img[0]),len(img)
    img1=crea(w,h)
    for x in range(w):
        for y in range(h):
            r,g,b=img[y][x]
            img1[y][x]=func(r,g,b)
    return img1
```

possiamo convertire un'immagine a colori in un'immagine in bianco e nero convertendo ciascuno dei tre canali alla media dei loro valori:

In [7]:

```
def grigio(r,g,b):
    grigio = (r + g + b) // 3
    return grigio, grigio, grigio
```

In [8]:

```
save(filtro(load('foto.png'),grigio), 'fotoBN.png')
Image('fotoBN.png')
```

Out[8]:

ESERCIZIO: progettare la funzione gradienteH(w, h, c0, c1) che disegna un gradiente di colore che va da sinistra a destra dal colore c0 al colore c1. Il disegno deve avere ampiezza w e altezza h.

In [9]:

```
def gradienteH(w, h, c0, c1):
    img = crea(w,h)
    for x in range(0, w):
        ratio = x/w
        cr = round(c0[0]*(1-ratio)+c1[0]*ratio)
        cg = round(c0[1]*(1-ratio)+c1[1]*ratio)
        cb = round(c0[2]*(1-ratio)+c1[2]*ratio)
        for y in range(0,h):
            img[y][x] = (cr, cg, cb)
    return img
```

In [10]:

```
rossoblu = gradienteH(700, 100, (255, 0, 0), (0, 0, 255))
save(rossoblu, 'rossoblu.png')
Image('rossoblu.png')
```

Out[10]:

ESERCIZIO: progettare la funzione gradienteV(w, h, c0, c1) che disegna un gradiente di colore che va dall'alto in basso dal colore c0 al colore c1. Il disegno deve avere ampiezza w e altezza h.

In [11]:

```
def gradienteV(w, h, c0, c1):
    img = crea(w,h)
    for y in range(0, h):
        ratio = y/h
        cr = round(c0[0]*(1-ratio)+c1[0]*ratio)
        cg = round(c0[1]*(1-ratio)+c1[1]*ratio)
        cb = round(c0[2]*(1-ratio)+c1[2]*ratio)
        for x in range(0,w):
            img[y][x] = (cr, cg, cb)
    return img
```

In [12]:

```
bianconero = gradienteV(200, 150, (255, 255, 255), (0, 0, 0))
save(bianconero, 'bianconero.png')
Image('bianconero.png')
```

Out[12]:

ESERCIZIO: progettare la funzione gradienteHV(w, h, c00, c01, c10, c11) che disegna un gradiente di colore combinato orizzontate-verticale che va da c00 in alto a sinistra, c01 in basso a sinistra, c10 in alto a destra, c11 in basso a destra. Il disegno deve avere ampiezza w e altezza h.

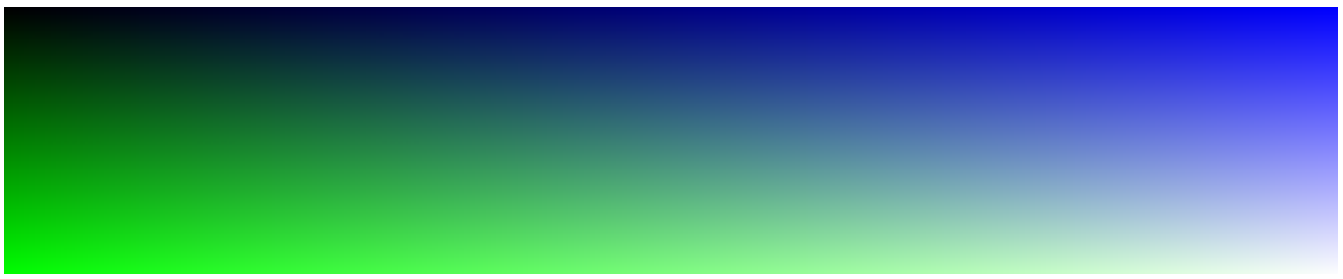
In [13]:

```
def gradienteHV(w, h, c00, c01, c10, c11):
    img = crea(w,h)
    for y in range(0, h):
        ratioy = y/h
        for x in range(0,w):
            ratiox = x/w
            colore = []
            for c in range(3):
                interpolato = round(
                            c00[c]*(1-ratioy)*(1-ratiox) +
                            c01[c]*ratioy*(1-ratiox) +
                            c10[c]*(1-ratioy)*ratiox +
                            c11[c]*ratioy*ratiox)
                colore+=[(interpolato)]
            img[y][x] = tuple(colore)
    return img
```

In [14]:

```
nero = (0, 0, 0)
verde = (0, 255, 0)
blu = (0, 0, 255)
bianco = (255, 255, 255)
misto = gradienteHV(500, 100, nero, verde, blu, bianco)
save(misto, 'misto.png')
Image('misto.png')
```

Out[14]:



ESERCIZIO: scrivere la funzione copia(sorgente, destinazione, x, y, w, h, x1, y1) che copia una porzione dell'immagine sorgente sull'immagine destinazione. La porzione è il rettangolo che ha il vertice in alto a sinistra nel pixel x,y ed ha ampiezza w e altezza h. Questo rettangolo va riportato nell'immagine destinazione a partire dal pixel x1,y1

In [15]:

```
def inside(img, x, y):
    return 0 <= x < len(img[0]) and 0 <= y < len(img)


def copia(sorgente, destinazione, x, y, w, h, x1, y1):
    for dx in range(w):
        for dy in range(h):
            xs = x + dx
            ys = y + dy
            xd = x1 + dx
            yd = y1 + dy
            if inside(sorgente, xs, ys) and inside(destinazione, xd, yd):
                destinazione[yd][xd] = sorgente[ys][xs]
```

In [16]:

```
sorgente= load('foto.png')
destinazione=load('foto.png')
destinazione=inverti(destinazione)
w, h = len(sorgente[0]), len(sorgente)
copia(sorgente, destinazione, 0, 0, w, h, w//4, round(h/2))
save(destinazione, 'copiamista.png')
Image('copiamista.png')
```

Out[16]:

ESERCIZIO: progettare una funzione mosaico1(img,s) che ritorna un'immagine ottenuta da img suddividendola in quadrati di lato s e riempiendo ogni quadrato con il colore del pixel in altro a sinistra del quadrato

In [17]:

```
def mosaico1(img,s):
    w, h = len(img[0]), len(img)
    img1 = crea(w, h)
    # itera sui possibili quadrati
    for i in range(h//s):
        for j in range(w//s):
            # colore dell'angolo in alto-sinistra
            c = img[i*s][j*s]
            disegna_quadrato(img1, j*s, i*s, s, c)
    return img1

def disegna_quadrato(img, x, y, s, c):
    # disegna su img un quadratino di lato s e colore c a partire dal pixel x,
y
    for px in range(x, x+s):
        for py in range(y, y+s):
            if inside(img,px,py):
                img[py][px] = c
```

In [18]:

```
save(mosaico1(load('foto.png'),5), 'mos1.png')
Image('mos1.png')
```

Out[18]:

ESERCIZIO: progettare una funzione mosaico2(img,s) che ritorna un'immagine ottenuta da img suddividendola in quadrati di lato s e riempiendo ogni quadrato con la media dei colori dei pixel presenti nel quadrato.

In [19]:

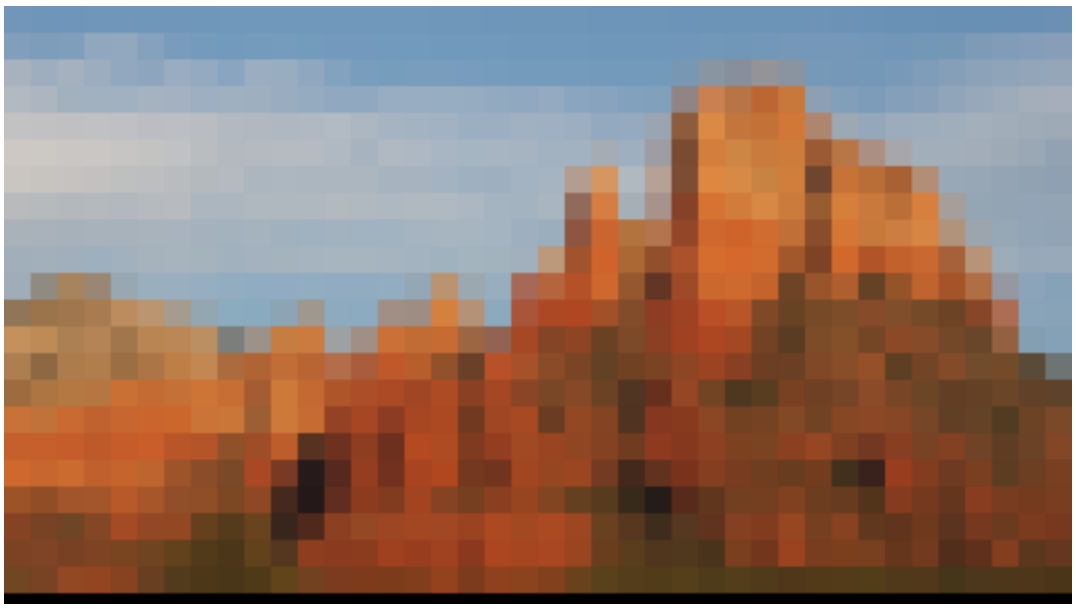
```
def mosaico2(img, s):
    w, h = len(img[0]), len(img)
    img1 = crea(w, h)
    # itera sui possibili quadratini
    for i in range(h//s):
        for j in range(w//s):
            # colore medio dell'immagine
            c = media(img,j*s,i*s,s)
            disegna_quadrato(img1, j*s, i*s, s, c)
    return img1

def media(img, i, j, s):
    '''Calcola la media dei valori nel quadratino di lato s che ha il pixel in
alto a
    sinistra nella posizione j,i'''
    c = [0,0,0]
    for x in range(j,j+s):
        for y in range(i,i+s):
            if inside(img, y, x):
                for k in range(3):
                    c[k] += img[x][y][k]
    for k in range(3):
        c[k] //= s*s
    return tuple(c)
```

In [20]:

```
save(mosaico2(load('foto.png'),10), 'mos2.png')
Image('mos2.png')
```

Out[20]:



ESERCIZIO: progettare una funzione scramble(img, d, s) che ritorna una nuova immagine ottenuta colorando ogni pixel (i, j) con il colore di un pixel scelto a caso nel quadratino centrato in (i, j) di lato 2*d + 1

In [21]:

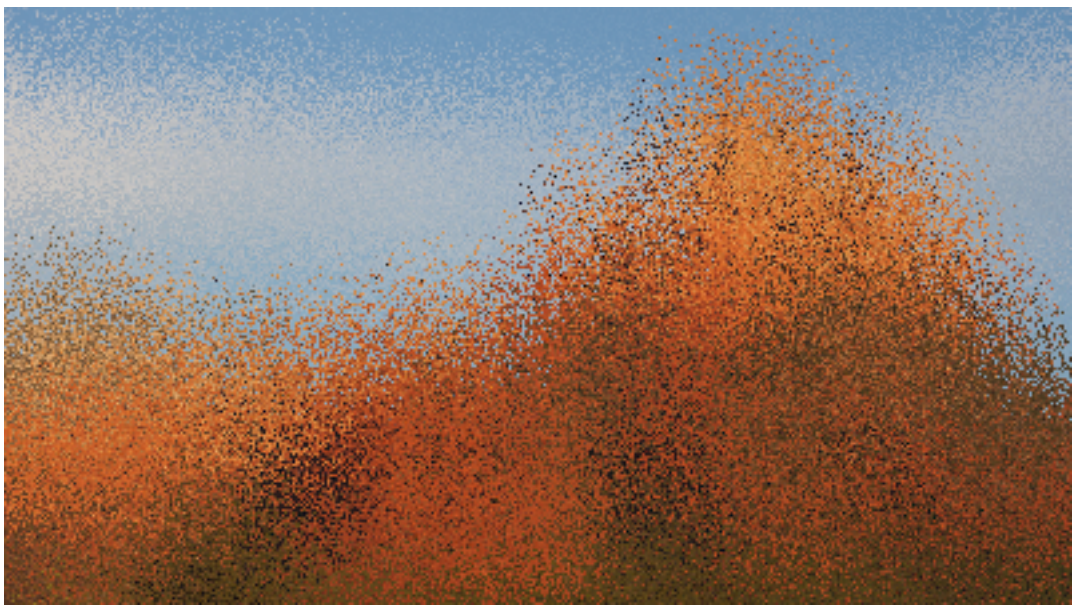
```
from  random import randint,seed

def mosaico3(img, d, s):
    # settiamo il seed per generare la stessa sequenza di numeri casuali
    seed(s)
    w, h = len(img[0]), len(img)
    img1 = crea(w, h)
    for j in range(h):
        for i in range(w):
             # sceglie a caso un pixel nel quadrato
            ri = i + randint(-d,d)
            rj = j + randint(-d,d)
            # evitando che si esca dall'immagine
            ri = max(0, min(w-1, ri))
            rj = max(0, min(h-1, rj))
            img1[j][i] = img[rj][ri]
    return img1
```

In [22]:

```
save(mosaico3(load('foto.png'),20,1), 'mos3.png')
Image('mos3.png')
```

Out[22]:



ESERCIZIO: progettare la funzione mini(img,s) che ritorna una nuova immagine di dimensioni ridotte (se img ha dimensioni w e h allora img1 avrà dimensioni w//s e h//s.

In [23]:

```
def mini1(img,s):
    w, h = len(img[0]), len(img)
    img1 = crea(w//s, h//s)
    # itera sui possibili pixel
    for i in range(h//s):
        for j in range(w//s):
            # colore del nuovo pixel
            img1[i][j] = img[i*s][j*s]
    return img1
```

In [24]:

```
save(mini1(load('foto.png'),2), 'mini.png')
Image('mini.png')
```

Out[24]:

In [25]:

```
def mini2(img,s):
    w, h = len(img[0]), len(img)
    img1 = crea(w//s, h//s)
    # itera sui possibili pixel
    for i in range(h//s):
        for j in range(w//s):
            # colore del nuovo pixel
            img1[i][j] = media(img, j*s, i*s, s)
    return img1
```

In [26]:

```
save(mini2(load('foto.png'),2), 'mini.png')
Image('mini.png')
```

Out[26]: