Table of Contents

Fondamenti di Programmazione

Andrea Sterbini

lezione 13 - 17 novembre 2022

RECAP: Immagini

- creazione/load/save
- rotazione
- disegno di linee verticali/orizzontali/diagonali
- disegno di rettangoli ed ellissi

In [1]:
```
%load_ext nb_mypy
```

```
Version 1.0.4
```

In [2]:
```
from images import load, visd

# definiamo qualche colore
black =   0,   0,   0
white = 255, 255, 255
red   = 255,   0,   0
green =   0, 255,   0
blue  =   0,   0, 255
cyan  =   0, 255, 255
yellow= 255, 255,   0
purple= 255,   0, 255
gray  = 128, 128, 128

# definizione di tipi
Colore   = tuple[int,int,int]
Immagine = list[list[Colore]]

def crea_immagine(larghezza : int, altezza : int, colore : Colore=black) -> Immagine
    return [
        [ colore ]*larghezza for i in range(altezza)
        ]

def draw_pixel(img : Immagine, x : int, y : int, colore : Colore) -> None:
    altezza   = len(img)
    larghezza = len(img[0])
    if  0 <= x < larghezza and 0 <= y < altezza:
        img[y][x] = colore
```

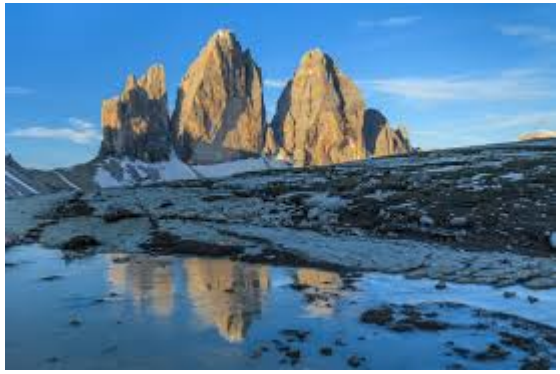
Ritagliare una immagine (crop)

In [3]:

```
# tolgo una striscia attorno all'immagine di spessori dati
def crop_image(img : Immagine,
               alto : int, sx : int,
               basso : int, dx :int) -> Immagine :
    # FIXME: aggiungere controllo sui parametri
    # copio solo il gruppo di righe giuste
    if basso > 0:
        fetta = img[alto:-basso]
        # se basso==0 bisogna scrivere così per arrivare in fondo sennò si copia da a
    else:
        fetta = img[alto:]
    if dx > 0:
        return [ riga[sx:-dx] for riga in fetta ]
        # per ciascuna riga copio solo la slice di colonne giusta
    else:
        return [ riga[sx:] for riga in fetta ]
        # se dx==0 bisogna scrivere così per arrivare in fondo sennò si copia da sx a
```

In [4]:

```
# carico la foto per gli esempi che seguono
img = load('3cime.png')
# esempio
cropped : Immagine = crop_image(img, 30, 20, 20, 50)
visd(img), visd(cropped)
None
```



Copia e incolla parte dell'immagine su un'altra

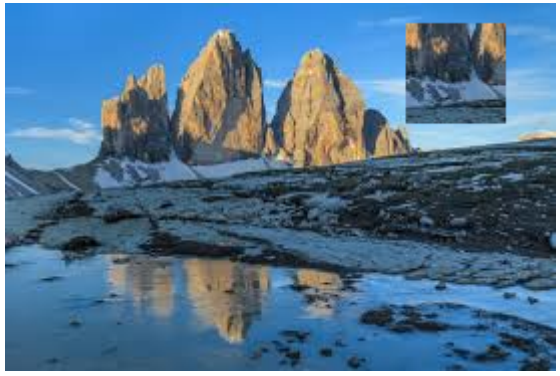
- con un crop
- e un paste
- con traslazione di coordinate

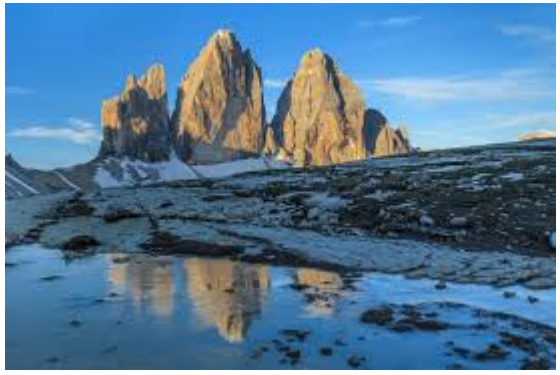
In [6]:

```python
# per copiare l'immagine (o una sua parte) in un'altra
def cut_paste_img(imgS : Immagine,
                  imgD : Immagine,
                  xs1 : int, ys1 : int, xs2 : int, ys2 : int,
                  XD : int, YD : int ) -> None:
    # FIXME: controllo sui parametri
    # ATTENZIONE: assumo che i parametri siano corretti
    HS = len(imgS)
    WS = len(imgS[0])
    # prima creo il frammento da copiare
    # FIXME: prima di ritagliare calcoliamo quante righe e quante colonne
    # entreranno nell'immagine di destinazione e ritagliamo solo quella parte
    frammento = crop_image(imgS, ys1, xs1, HS-ys2, WS-xs2 )
    # per tutte le righe da copiare
    larghezza = len(frammento[0])
    for yF,riga in enumerate(frammento):
    # uso un assegnamento a slice
        imgD[yF+YD][XD:XD+larghezza] = riga
```

In [9]:

```
img=load('3cime.png')

# copio l'immagine delle 3 cime di Lavaredo
img_copiata = crop_image(img, 0,0,0,0)

# altro modo di copiare una immagine
def copia(img: Immagine) -> Immagine:
    return [ riga.copy() for riga in img ]   # NON va bene usare copy sulla lista est

# LAVAGNA!

img_copiata = copia(img)

# ne copio un pezzettino in un altro punto
cut_paste_img(img, img_copiata, 50,50,100,100, 200,10 )
visd(img_copiata), visd(img)
```





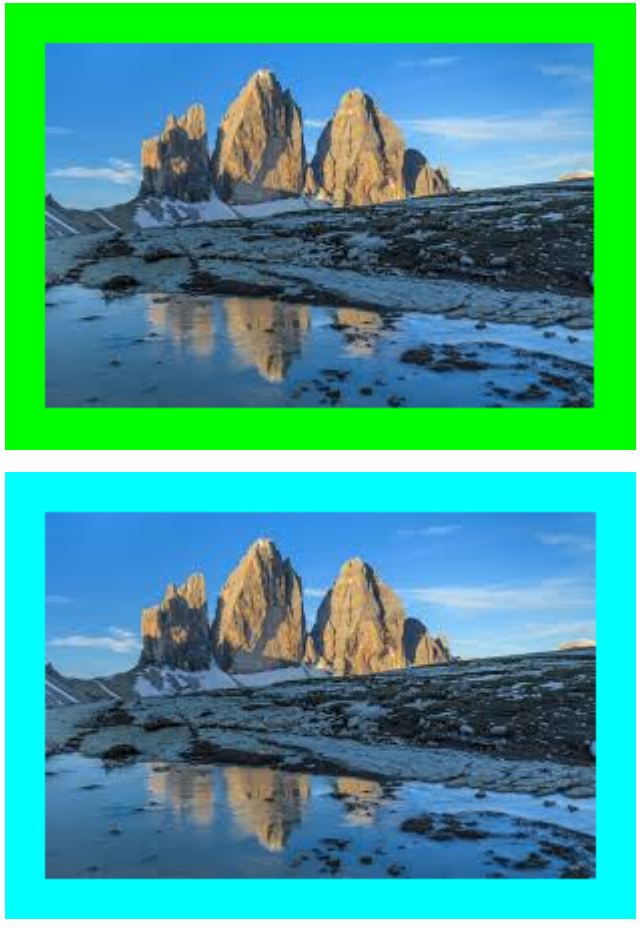
Out[9]: (None, None)

Aggiungere un bordo

In [14]:

```python
# per aggiungere un bordo
def add_border(img : Immagine,
               spessore : int, colore : Colore ) -> Immagine :
    L, A = len(img[0]), len(img)
    # creiamo una immagine più grande col colore del bordo
    nuova = crea_immagine(L+2*spessore,A+2*spessore, colore)
    # ci incolliamo l'immagine originale
    cut_paste_img(img,nuova,0,0,L-1,A-1,spessore,spessore)
    return nuova

# oppure la costruiamo riga per riga
def add_border2(img : Immagine, spessore : int, colore : Colore ) -> Immagine :
    L, A = len(img[0]), len(img)
    bordata = []
    # - spessore righe del colore
    bordata += [ [colore] * (L+2*spessore)
                 for i in range(spessore)  ]
            # - per ogni riga dell'immagine
    for riga in img:
        bordata.append( [colore]*spessore + riga + [colore]*spessore  )
            #   - spessore pixel + riga + spessore pixel
            # - spessore righe del colore
    bordata += [ [colore] * (L+2*spessore)
                 for i in range(spessore)  ]
    return bordata

```

In [15]:
```
1 bordata = add_border(img, 20, green)
2 bordata2 = add_border2(img, 20, cyan)
3 visd(bordata) , visd(bordata2)
```



Out[15]: (None, None)

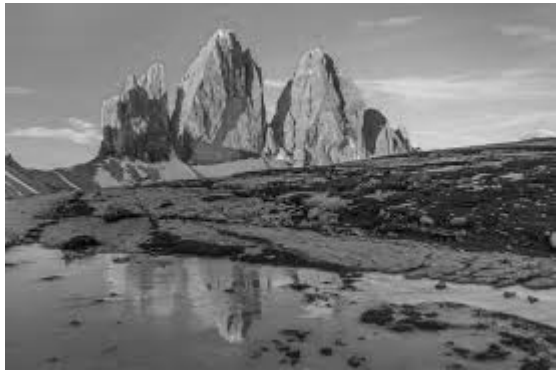
Filtri da applicare ai colori

- ogni pixel della immagine viene trasformato. Esempi
  - toni di grigio
  - negativo
  - incremento/riduzione della luminosità
  - incremento/riduzione del contrasto

NOTA: questi filtri dipendono solo dal pixel in esame

toni di grigio

In [16]:
```python
# filtro grigio che trasforma un colore in grigio con la stessa luminositàù
def filtro_grigio(colore : Colore) -> Colore :
    # tutti i pixel devono essere grigi ma con la stessa luminosità totale
    # ovvero R=G=B e R+G+B uguale a prima, quindi bisogna mediare
    media = sum(colore)//3
    return media, media, media

# per trasformare una immagine in livelli di grigio
def grey(img : Immagine) -> Immagine :
    # la copio
    grigia = copia(img)
        # e sostituisco ogni pixel col grigio corrispondente
    for y, riga in enumerate(img):
        for x, pixel in enumerate(riga):
            grigia[y][x] = filtro_grigio(pixel)
    return grigia

# esempio
img_grigia = grey(img)
visd(img_grigia)
```



luminosità

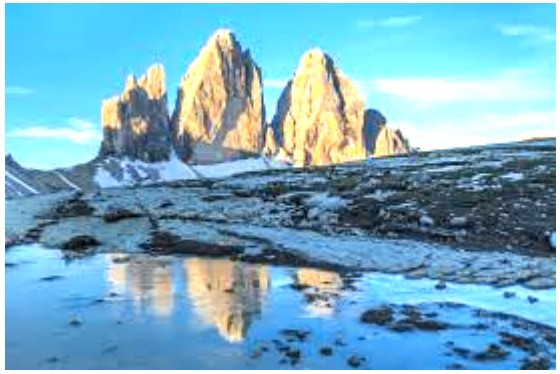
Amplifichiamo/riduciamo di **k** la luminosità dell'immagine

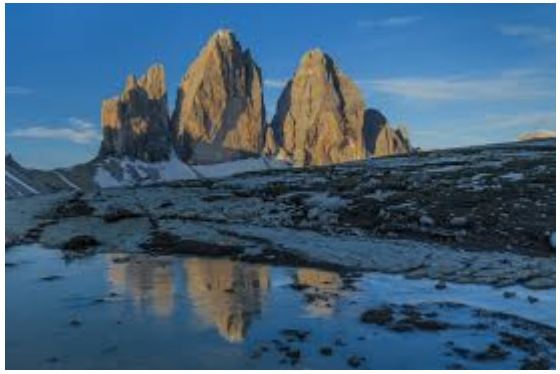
In [17]:

```python
from copy import deepcopy

# per schiarire/scurire una immagine di un fattore k (float)
def luminosità(img : Immagine, k : float) -> Immagine:
    # creo una nuova immagine con deepcopy
    copia = deepcopy(img)
    # tutti i pixel devono avere una luminosità moltiplicata per k
    for y, riga in enumerate(img):
        for x, colore in enumerate(riga):
            copia[y][x] = filtro_lumi(colore, k)  # sostituisco il pixel
    return copia

def filtro_lumi(colore : Colore, k : float) -> Colore:
    R,G,B = colore
    # mi assicuro che i valori risultanti siano interi nel range 0..255
    return bound(R*k), bound(G*k), bound(B*k)

# Ci conviene definire una funzione che vincola il risultato
# ad essere INTERO ed entro un dato INTERVALLO  [m,M] compresi
def bound(canale : float|int, m:int=0, M:int=255 ) -> int:
    # trasformo il valore in intero all'interno di [m..M]
    canale = int(round(canale))
    return min(max(canale, m), M)
```

In [18]:

```
# esempio
img_luminosa = luminosità(img, 1.5)
img_scura    = luminosità(img, 0.8)

visd(img_luminosa), visd(img_scura)
None
```





Generalizziamo l'applicazione del filtro

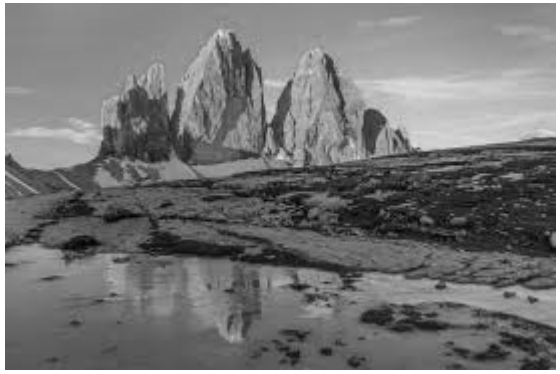
- definendo una trasformazione generica
- che accetta come parametro la funzione che trasforma il pixel

In [21]:

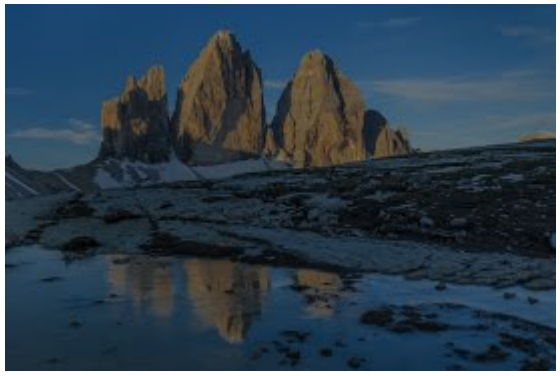
```python
from typing import Callable

def applica_filtro( img : Immagine,
                    filtro : Callable[[Colore], Colore] ) -> Immagine:
    # passo nell'argomento 'filtro' una funzione che calcola
    # per ogni colore il nuovo colore
    # copio l'immagine
    copia = deepcopy(img)
    # tutti i pixel vengono sostituiti con il risultato del filtro
    for y, riga in enumerate(img):
        for x, colore in enumerate(riga):
            copia[y][x] = filtro(colore)   ### QUI chiamo il filtro
    return copia
```

In [22]:

```python
# esempio
img_ingrigita = applica_filtro(img, filtro_grigio)
visd(img_ingrigita)
```



In [27]:
```python
# il filtro deve accettare un solo parametro
def piu_scura(pixel):
    return filtro_lumi(pixel, 0.5)
scura = applica_filtro(img, piu_scura)

# oppure posso usare una lambda
scura = applica_filtro(img,
                       lambda pixel: filtro_lumi(pixel, 0.5))
visd(scura)
```



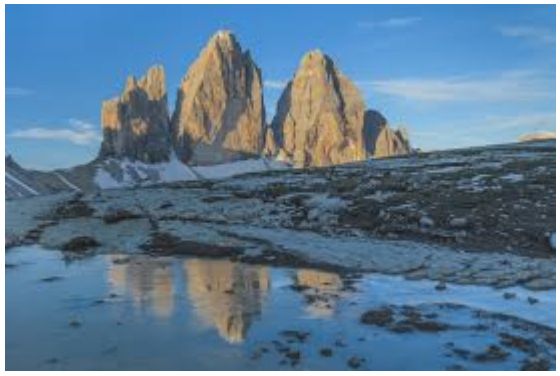
Contrasto

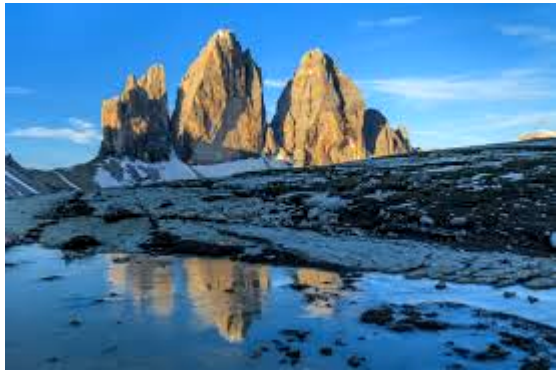
- per cambiare il contrasto di un fattore **k**
  - ogni pixel chiaro deve diventare più chiaro
  - ogni pixel scuro deve diventare più scuro
  - ovvero si devono allontanare/avvicinare di un fattore **k** dal grigio **128,128,128**

In [28]:
```python
def filtro_contrasto(colore : Colore, k : float):
    # aumento di un fattore k la distanza del colore da 128
    return tuple( bound((componente-128)*k+128) for componente in colore)

# PROBLEMA! : filtro_contrasto vuole DUE parametri
```

In [29]:

```
# SOLUZIONE : definisco una lambda che aggiunge K
# esempio
img_più_contrastata  = applica_filtro(img,
                       lambda colore: filtro_contrasto(colore, 1.2))
img_meno_contrastata = applica_filtro(img,
                       lambda colore: filtro_contrasto(colore, 0.8))

visd(img_meno_contrastata), visd(img_più_contrastata)
None
```

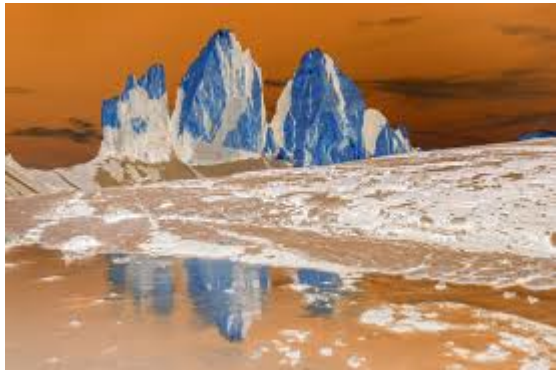




effetto Negativo

In [31]:

```python
# inverto la luminosità
def negativo(colore : Colore ) -> Colore :
    return tuple( 255-componente for componente in colore )

img_negata = applica_filtro(img, negativo)
visd(img_negata)
```



Sfocatura (blur)

- mediamo i colori fino a distanza **k** dal pixel

In [32]:

```python
# per sfocare una immagine entro una distanza k
    # genero una nuova immagine
    # con i pixel che sono la media del gruppo di pixel
    # attorno a quello indicato fino a distanza k
def blur(img : Immagine, k : int) -> Immagine:
    W = len(img[0])
    H = len(img)
    copia = [ riga.copy()  for riga in img ] # invece che deepcopy
    for x  in range(W):
        for y in range(H):
            # raccolgo i colori del vicinato
            vicinato = []
            for X in range(x-k,x+k+1):
                for Y in range(y-k, y+k+1):
                    if 0 <= X < W and 0 <= Y < H:
                        vicinato.append(img[Y][X])
            copia[y][x] = colore_medio(vicinato)
    return copia
# blur è una operazione molto lenta ....

# TODO: realizzarlo come filtro che dipende dalla posizione -> vedi sotto
```

```
<cell>17: error: Name "colore_medio" is not defined  [name-defined]
```

In [33]:

```python
# calcolo la media di un gruppo di colori
def colore_medio(listaColori : list[Colore]) -> Colore :
    N     = len(listaColori)
    R,G,B = 0, 0, 0
    for r,g,b in listaColori:
        R += r
        G += g
        B += b
    return bound(R/N), bound(G/N), bound(B/N)

#oppure
#    return tuple(map(lambda X: bound(sum(X)/N), zip(*listaColori)))
```

In [34]:
```
# esempio
img_sfocata1 = blur(img, 1)
img_sfocata2 = blur(img, 2)
img_sfocata3 = blur(img, 3)
visd(img_sfocata1), visd(img_sfocata2), visd(img_sfocata3)
None
```

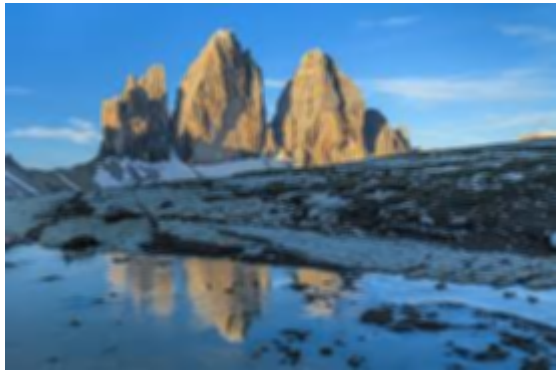


Image noise

In [35]:
```
from random import randint

# per aggiungere rumore casuale ad una immagine
    # possiamo aggiungere a ciascun pixel un piccolo valore random
def rumore_casuale(colore : Colore, k : int) -> Colore:
    return tuple( bound(C + randint(-k,k)) for C in colore )
```

```
<cell>6: error: Incompatible return value type (got "Tuple[int, ...]", expected "Tuple[in
t, int, int]")  [return-value]
```

In [35]:

```
# esempio
poco_rumore = applica_filtro(img, lambda C: rumore_casuale(C, 20))
tanto_rumore = applica_filtro(img, lambda C: rumore_casuale(C, 50))
visd(poco_rumore), visd(tanto_rumore)
None
```

Filtri che dipendono dalla posizione

Abbiamo bisogno di filtri che conoscono:

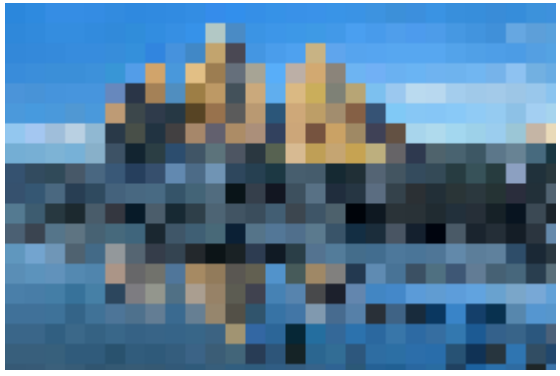
- la posizione **x,y** del pixel corrente
- l'immagine sorgente (per leggere altri pixel)
- le dimensioni dell'immagine (per evitare di ricalcolarle)

Esempio: Pixellazione

- coloro il pixel corrente come il centro del suo quadratino
- oppure come la media del suo quadratino

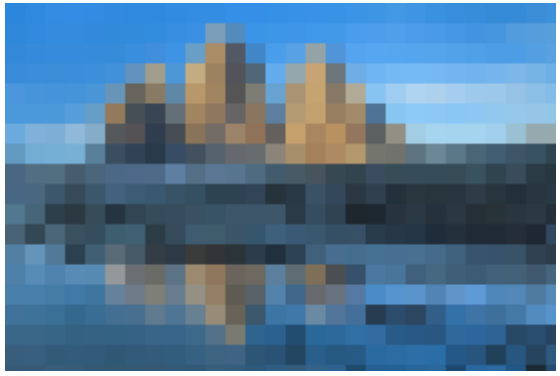
In [37]:
```
# - devo sapere dove sono nella immagine e avere accesso a tutta l'immagine!
def applica_filtro_XY( img : Immagine,
                       filtro : Callable[[int, int, Immagine, int, int],
                                         Colore] ) -> Immagine:
    W,H = len(img[0]),len(img)
    # passo nell'argomento 'filtro' una funzione che calcola
    # per ogni colore e posizione X,Y il nuovo colore
    copia = deepcopy(img)
    # tutti i pixel che devono avere una luminosità moltiplicata per k
    for y in range(H):
        for x in range(W):
            copia[y][x] = filtro(x, y, img, W, H)   ### QUI chiamo il filtro
    return copia
```

In [38]:
```
# ad ogni quadrato sostituiamo il colore del suo centro
def pixella(x : int, y : int, img : Immagine, W : int, H : int, S : int) -> Colore :
    X = bound(x-x%S+S/2, 0, W-1) # X del centro
    Y = bound(y-y%S+S/2, 0, H-1) # Y del centro
    return img[Y][X]

pixellata = applica_filtro_XY(img,
            lambda x,y,imm,W,H: pixella(x,y,imm,W,H,10))
visd(pixellata)
```



In [39]:

```python
# ad ogni quadrato sostituiamo la *media* dei colori
def pixelmedio(x : int, y : int, img : Immagine, W : int, H : int, S : int) -> Colore
    R,G,B, N = 0,0,0, 0
    minx = x-x%S
    miny = y-y%S
    for X in range(minx, min(W,minx+S)):
        for Y in range(miny, min(H,miny+S)):
            r,g,b = img[Y][X]
            R += r
            G += g
            B += b
            N += 1                      # conto i soli pixel sommati
    return R//N, G//N, B//N

## INEFFICIENTE: ricalcola la media per ogni pixel
## MEGLIO: calcolo la media una volta per ogni quadrato
# - ad esempio ricordando il risultato per ogni xmin,ymin,xmax,ymax

pixellata2 = applica_filtro_XY(img, lambda x,y,imm,W,H: pixelmedio(x,y,imm,W,H,10))
visd(pixellata2)
```



Blur come filtro

- per ogni pixel calcolo la media del vicinato

In [40]:
```
def blur_filter(x : int, y : int, img : Immagine, W : int, H : int, k : int) -> Colore
    # raccolgo i vicini fino a distanza k
    vicini = []
    for X in range(bound(x-k,0,W),bound(x+k+1, 0, W)):
        for Y in range(bound(y-k,0,H),bound(y+k+1, 0, H)):
            vicini.append(img[Y][X])
    # ne torno la media
    return colore_medio(vicini)
```

In [41]:
```
# Esempio
sfumata = applica_filtro_XY(img, lambda x,y,imm,W,H: blur_filter(x,y,imm,W,H, 3 ))
visd(sfumata)
```

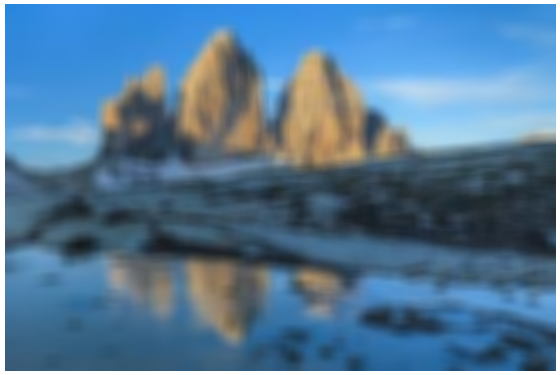


immagine rumorosa per spostamento di pixels

- scegliamo a caso un pixel antro una distanza **k**

In [42]:

```python
# per aggiungere rumore casuale ad una immagine
    # oppure sostituiamo ciascun pixel con un suo vicino preso a caso
def scegli_vicino_a_caso(x : int, y : int, img : Immagine, W : int, H : int, k : int)
    dx = randint(-k, k)
    dy = randint(-k, k)
    X = bound(x+dx, 0, W-1)
    Y = bound(y+dy, 0, H-1)
    return img[Y][X]

rumore = applica_filtro_XY(img, lambda x, y, imm, W, H: scegli_vicino_a_caso(x, y, im
visd(rumore)
```

Lente di ingrandimento

In [43]:

```python
from math import dist
# per dare l'effetto lente
    # nella zona della lente
    # fino a un raggio r
    # mettiamo dei pixel che stanno a distanza K volte
    # la loro distanza dal centro x1,y1 della lente

def lente(x : int, y : int, img : Immagine, W : int, H : int,
          x1 : int, y1 : int, r : int, k : float) -> Colore:
    D = dist((x,y), (x1,y1))
    if D > r:
        return img[y][x]
    # altrimenti amplifichiamo dx e dy di un fattore k
    dx = (x-x1)*k
    dy = (y-y1)*k
    # ci assicuriamo di essere nella immagine
    X = bound(x1+dx,0,W-1)
    Y = bound(y1+dy,0,H-1)
    return img[Y][X]

## LAVAGNA!
```

In [29]:

```
# esempio
ingrandita    = applica_filtro_XY(img,
    lambda x, y, img, W, H: lente(x, y, img, W, H, 100, 100, 100, 0.5) )
rimpicciolita = applica_filtro_XY(img,
    lambda x, y, img, W, H: lente(x, y, img, W, H, 100, 100, 100, 2   ) )
visd(ingrandita), visd(rimpicciolita)
None
```