Code for 4th :

```
from PIL import Image
import numpy as np
import math, os
import matplotlib.pyplot as plt

# ---------------- User Input ----------------
image_path = "C:/Users/dhivy/OneDrive/Desktop/sem- 6/DIP/5th question/Torgya - Arunachal
Festival.jpg"
# ---------------- Helper Functions ----------------

def load_image(path):
    img = Image.open(path).convert("RGB")
    return np.array(img, dtype=np.float32)

def resize_small(im, width=800):
    h, w = im.shape[:2]
    scale = width / w
    new_h = int(h * scale)
    img_pil = Image.fromarray(im.astype(np.uint8))
    img_resized = img_pil.resize((width, new_h), Image.BILINEAR)
    return np.array(img_resized, dtype=np.float32)

def show_inline(img, title=None):
    plt.figure(figsize=(8,5))
    plt.imshow(img.astype(np.uint8))
    plt.axis("off")
    if title:
        plt.title(title)
    plt.show()

# -------- BOX FILTER IMPLEMENTATION (NUMPY) --------
def box_filter(img, ksize, normalize=True):
    pad = ksize // 2
    img_pad = np.pad(img, ((pad,pad),(pad,pad),(0,0)), mode='edge')
    output = np.zeros_like(img)

    kernel = np.ones((ksize, ksize), dtype=np.float32)
    if normalize:
        kernel /= (ksize * ksize)

    for y in range(img.shape[0]):
        for x in range(img.shape[1]):
            region = img_pad[y:y+ksize, x:x+ksize]
            output[y, x] = np.sum(region * kernel[:, :, None], axis=(0,1))

    return output
```

```
# -------- GAUSSIAN FILTER IMPLEMENTATION (NUMPY) --------
def compute_sigma_from_size(size):
    return size / 6.0

def kernel_size_from_sigma(sigma):
    k = int(2 * math.ceil(3 * sigma) + 1)
    return max(k, 3)

def gaussian_1d(sigma, normalize=True):
    k = kernel_size_from_sigma(sigma)
    r = k // 2
    x = np.arange(-r, r+1, dtype=np.float64)
    g = np.exp(-(x**2) / (2 * sigma * sigma))
    if normalize:
        g = g / g.sum()
    return g.astype(np.float32)

def separable_gaussian(img, sigma, normalize=True):
    g = gaussian_1d(sigma, normalize=normalize)
    k = len(g)
    pad = k // 2

    # Pad image
    img_pad = np.pad(img, ((pad,pad),(pad,pad),(0,0)), mode='edge')

    # Horizontal pass
    temp = np.zeros_like(img)
    for y in range(img.shape[0]):
        for x in range(img.shape[1]):
            region = img_pad[y+pad, x:x+k]
            temp[y,x] = np.sum(region * g[:,None], axis=0)

    # Pad temp for vertical pass
    temp_pad = np.pad(temp, ((pad,pad),(0,0),(0,0)), mode='edge')

    output = np.zeros_like(img)
    for y in range(img.shape[0]):
        for x in range(img.shape[1]):
            region = temp_pad[y:y+k, x]
            output[y,x] = np.sum(region * g[:,None], axis=0)

    return output

# ---------------- Load & Resize Image ----------------
img = load_image(image_path)
img = resize_small(img, width=800)
```

```
show_inline(img, "Loaded & Resized Image")

# ---------- Create Output Folder ----------
out_dir = "results_no_cv2"
os.makedirs(out_dir, exist_ok=True)

outputs = {}

# ---------- BOX FILTERS ----------
for k in (5, 20):
    outputs[f"box_{k}_unnormalized"] = box_filter(img, k, normalize=False)
    outputs[f"box_{k}_normalized"]   = box_filter(img, k, normalize=True)

# ---------- GAUSSIAN FILTERS ----------
for size_hint in (5, 20):
    sigma = compute_sigma_from_size(size_hint)
    outputs[f"gaussian_{size_hint}_normalized"]   = separable_gaussian(img, sigma, normalize=True)
    outputs[f"gaussian_{size_hint}_unnormalized"] = separable_gaussian(img, sigma, normalize=False)
    print(f"For size {size_hint}: sigma={sigma:.3f}, kernel size={kernel_size_from_sigma(sigma)}")

# ---------- SAVE & DISPLAY OUTPUTS ----------
print("\nSaving output images...\n")
for name, im in outputs.items():
    save_path = os.path.join(out_dir, name + ".png")
    Image.fromarray(np.clip(im,0,255).astype(np.uint8)).save(save_path)
    print("Saved:", save_path)
    show_inline(im, title=name)

print("All processing completed successfully!")
```