Computer Vision Lab 1 - Basic image operations

Imports & name-adding function

In [1]:
```python
import cv2
import numpy as np

NAME = "Rens Delaplace"

def add_name_to_image(image):
    """
    Voeg de naam toe aan de afbeelding op een standaard positie (rechtsonder).
    :param image: De afbeelding waarop de naam wordt toegevoegd.
    :return: De afbeelding met de naam toegevoegd.
    """
    font = cv2.FONT_HERSHEY_SIMPLEX
    font_scale = 0.6
    thickness = 1

    # Bereken de grootte van de tekst
    text_size = cv2.getTextSize(NAME, font, font_scale, thickness)[0]

    # Bepaal de positie rechtsonder
    position = (image.shape[1] - text_size[0] - 10, image.shape[0] - 10)

    # Voeg de naam toe aan de afbeelding
    cv2.putText(image, NAME, position, font, font_scale, (255, 255, 255), thickn

    return image
```

Reading, manipulating and writing pixel data

Exercise 1

In [2]:
```python
# Read the image
image = cv2.imread("img/clouds.jpg")

# dimensions
height, width, channels = image.shape
print(f"Image dimensions: Height={height}, Width={width}, Channels={channels}")
```

```
Image dimensions: Height=360, Width=801, Channels=3
```

Question 1

> What do the dimensions of the image array represent?

They represent the hight, width and color channels of the image.

Assignment 1

> Crop the image so it becomes square by chopping off the a part on the right side.

In [3]:
```python
square_size = min(height, width)
cropped_image = image[:, :square_size]

cropped_image = add_name_to_image(cropped_image)
cv2.imwrite('out/assignement1.jpg', cropped_image)
```

Out[3]:
```
True
```

Assignment 2

> Discolor the image by reducing the intensity of the red value of every pixel by half.

In [4]:
```python
# Assignment 2: Reduce the red intensity by half
red_reduced = image.copy()
red_reduced[:, :, 2] = red_reduced[:, :, 2] // 2  # OpenCV uses BGR format
red_reduced = add_name_to_image(red_reduced)
cv2.imwrite('out/assignement2.jpg', red_reduced)
```

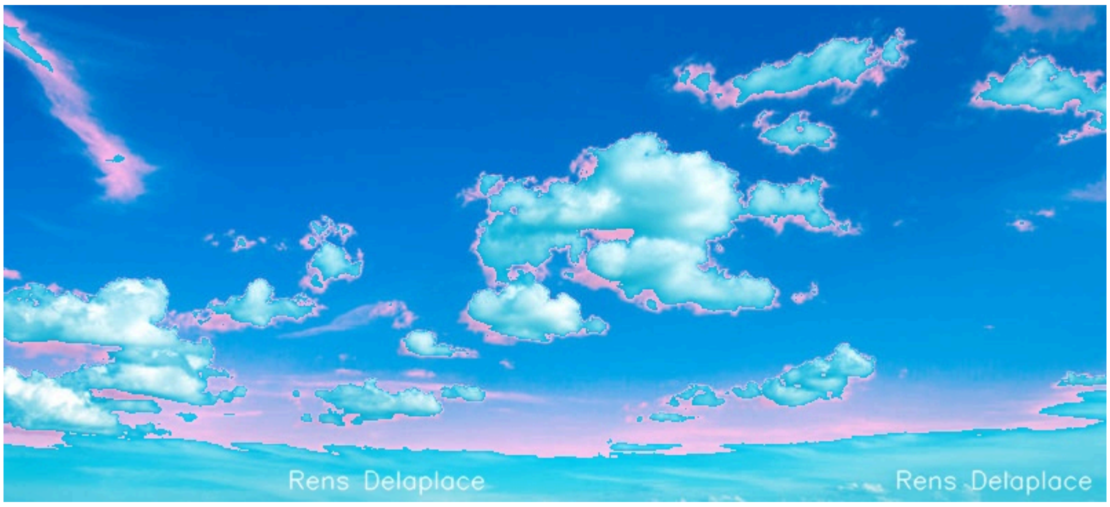
Out[4]:
```
True
```

Assignment 3

> Discolor the image by doubling the intensity of the red value of every pixel. You may have to handle an overflow problem (and use two more lines of code).

In [5]:
```
red_doubled = image.copy()
red_doubled[:, :, 2] = np.clip(red_doubled[:, :, 2] * 2, 0, 255)

red_doubled = add_name_to_image(red_doubled)
cv2.imwrite('out/assignement3.jpg', red_doubled)
```

Out[5]:
```
True
```

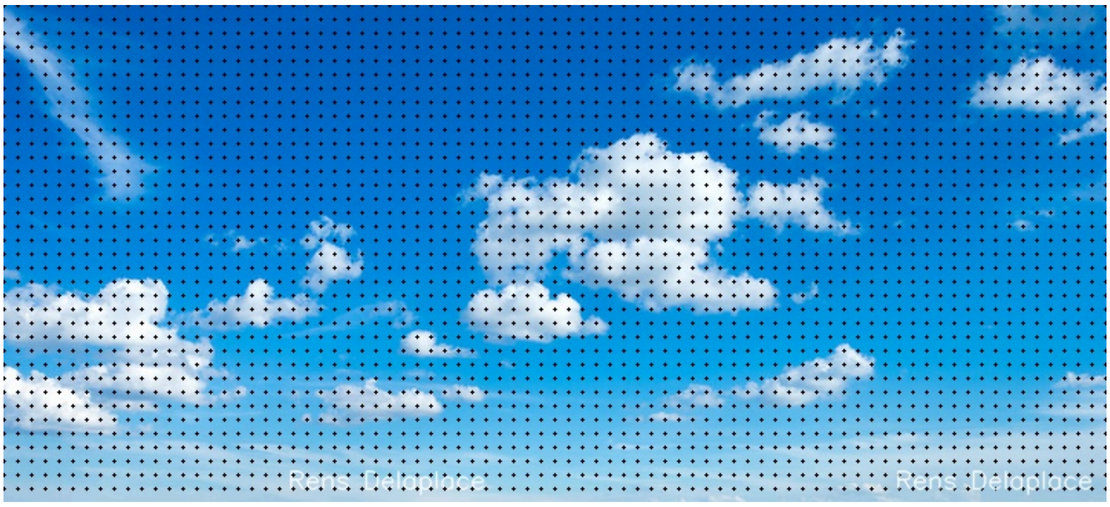


Assignment 4

> Make a regular grid of black dots on the image so that the dots are 10 pixels apart vertically and horizontally.

In [6]:
```
grid_image = image.copy()
for y in range(0, height, 10):
    for x in range(0, width, 10):
        cv2.circle(grid_image, (x, y), 1, (0, 0, 0), -1)  # Black dot
        # (grid_image → The image on which to draw
        # (x, y) → The position of the center of the circle.
        # 2 → Radius of the circle (2 pixels)
        # (0, 0, 0) → Black color in BGR format.
        # -1 → Fills the circle completely.

grid_image = add_name_to_image(grid_image)
cv2.imwrite('out/assignement4.jpg', grid_image)
```

Out[6]:
```
True
```



Tresholding

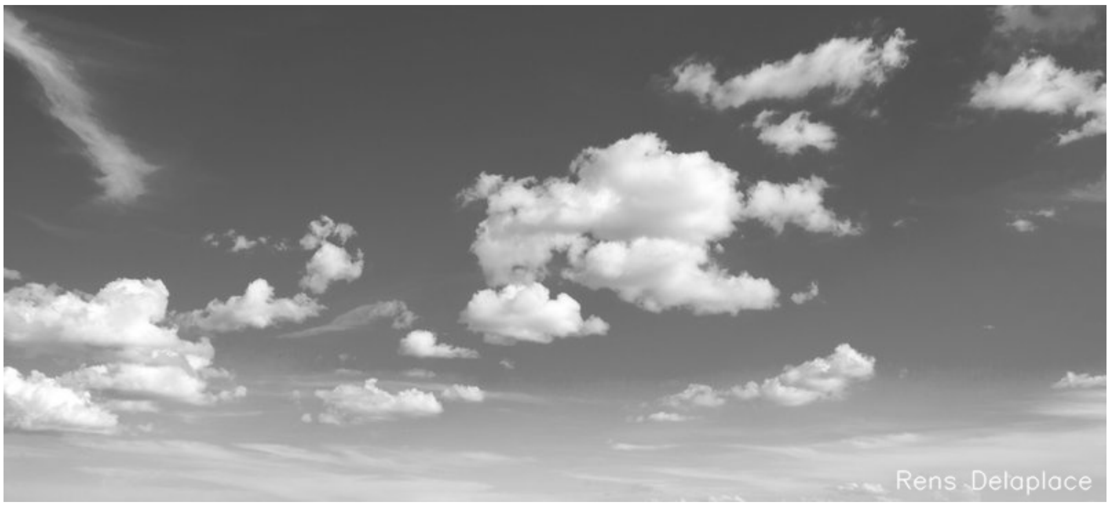
Exercise 2

Assignment 5

> Convert the image to a grayscale image.

In [7]:
```
image = cv2.imread('img/clouds.jpg')
grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

grayscale_image = add_name_to_image(grayscale_image)
cv2.imwrite('out/assignement5.jpg', grayscale_image)
```

Out[7]:
```
True
```



Assignment 6

> Threshold the grayscale image at 50% of the maximum value for this datatype.

In [8]:
```
# Get the maximum pixel value for the datatype
max_value = np.iinfo(image.dtype).max
threshold_value = max_value // 2  # 50% of max value

_, thresholded_image = cv2.threshold(grayscale_image, threshold_value, 255, cv2.
thresholded_image = add_name_to_image(thresholded_image)
cv2.imwrite('out/assignement6.jpg', thresholded_image)
```

Out[8]:
```
True
```

Assignment 7

> Threshold the grayscale image at the ideal threshold determined by Otsu's method.

In [9]:
```
_, otsu_thresholded_image = cv2.threshold(grayscale_image, 0, 255, cv2.THRESH_BI

otsu_thresholded_image = add_name_to_image(otsu_thresholded_image)
cv2.imwrite('out/assignement7.jpg', otsu_thresholded_image)
```

Out[9]:
```
True
```

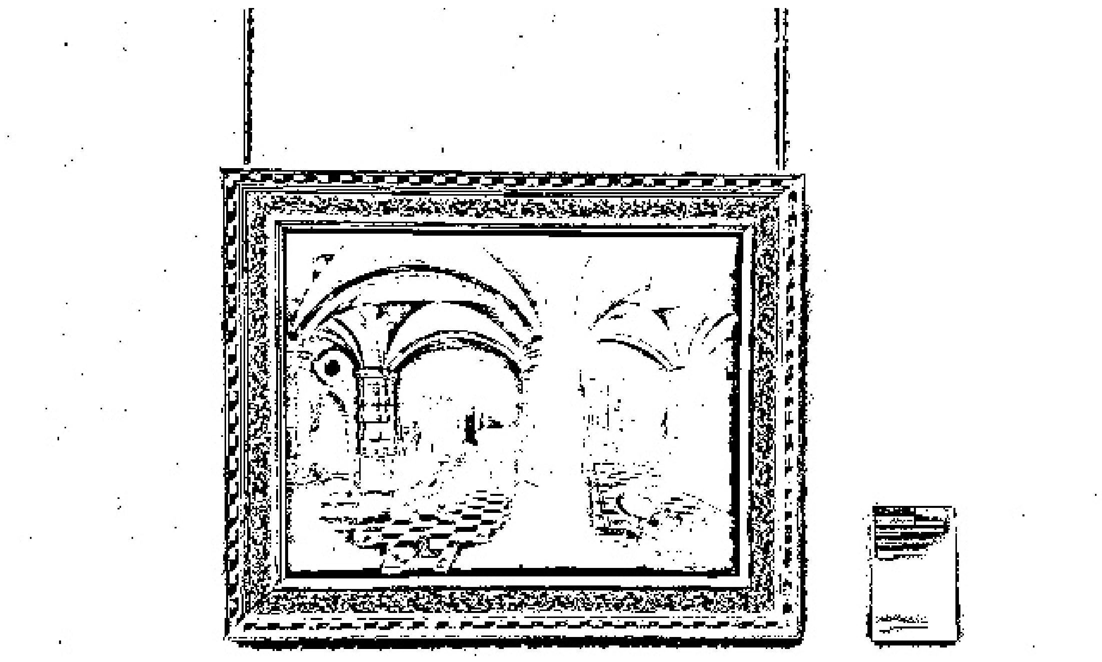
Exercise 3

Assignment 8

> Adaptively threshold the grayscale version of painting2.jpg so you get a similar result to the one below, where the background is uniformly white and you can cut out the painting along black lines.

In [10]:
```python
# Read the image in grayscale
painting_image = cv2.imread("img/painting2.jpg", cv2.IMREAD_GRAYSCALE)

# Ensure the image is in 8-bit format
painting_image = cv2.convertScaleAbs(painting_image)

adaptive_thresholded_image = cv2.adaptiveThreshold(painting_image, 255, cv2.ADAP

adaptive_thresholded_image = add_name_to_image(adaptive_thresholded_image)
cv2.imwrite("out/assignement8.jpg", adaptive_thresholded_image)
```

Out[10]:
```
True
```



Filtering

Exercise 4

Assignment 9

> Remove the white noise from whitenoise.png by Gaussian filtering. Find parameters for the Gaussian kernel that you find strike a good balance between noise level and blurriness of the result. This is subjective, but experiment with it!

In [11]:
```python
whitenoise_image = cv2.imread('img/whitenoise.png')

kernel_size = (7, 7)
sigma = 4
gaussian_filtered_image = cv2.GaussianBlur(whitenoise_image, kernel_size, sigma)

gaussian_filtered_image = add_name_to_image(gaussian_filtered_image)
cv2.imwrite('out/assignement9.jpg', gaussian_filtered_image)
```

Out[11]:
```
True
```

Question 2

> Can you choose the kernel size and sigma of the distribution independent of each other?

The kernel size (the size of the filter window) and sigma (the standard deviation of the Gaussian) are related.

If the kernel size is too small relative to sigma, important parts of the Gaussian distribution will be cut off, reducing accuracy. If the kernel size is too large, it increases computation time without significantly improving the result. Kernel size is often chosen as a multiple of sigma, for example 3*sigma. A larger kernel size generally requires a larger sigma for effective smoothing. However, they can be adjusted independently to find the right balance.

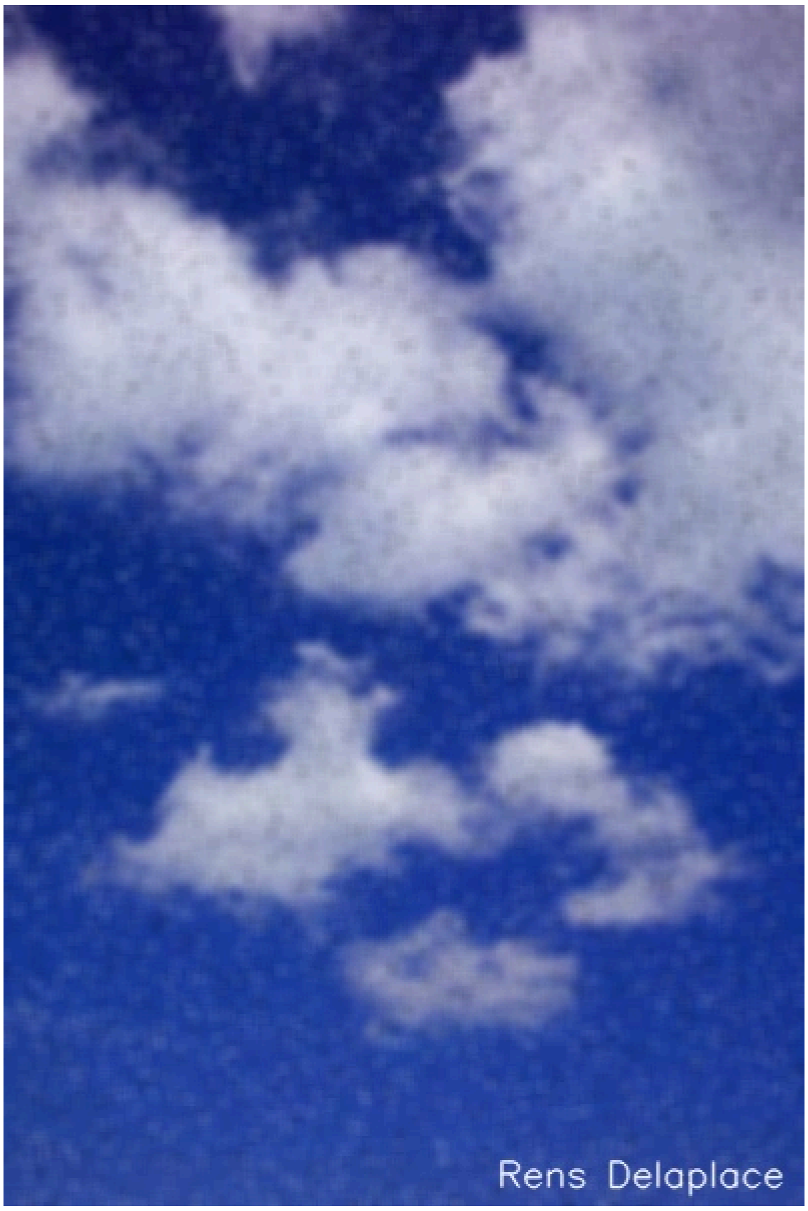
Exercise 5

Assignment 10

> Test the Gaussian filter on saltandpeppernoise.png.

In [12]:
```
saltpeppernoise_image = cv2.imread('img/saltandpeppernoise.png')

kernel_size = (7, 7)
sigma = 4
gaussian_filtered_image = cv2.GaussianBlur(saltpeppernoise_image, kernel_size, s

gaussian_filtered_image = add_name_to_image(gaussian_filtered_image)
cv2.imwrite('out/assignement10.jpg', gaussian_filtered_image)
```

Out[12]:
```
True
```

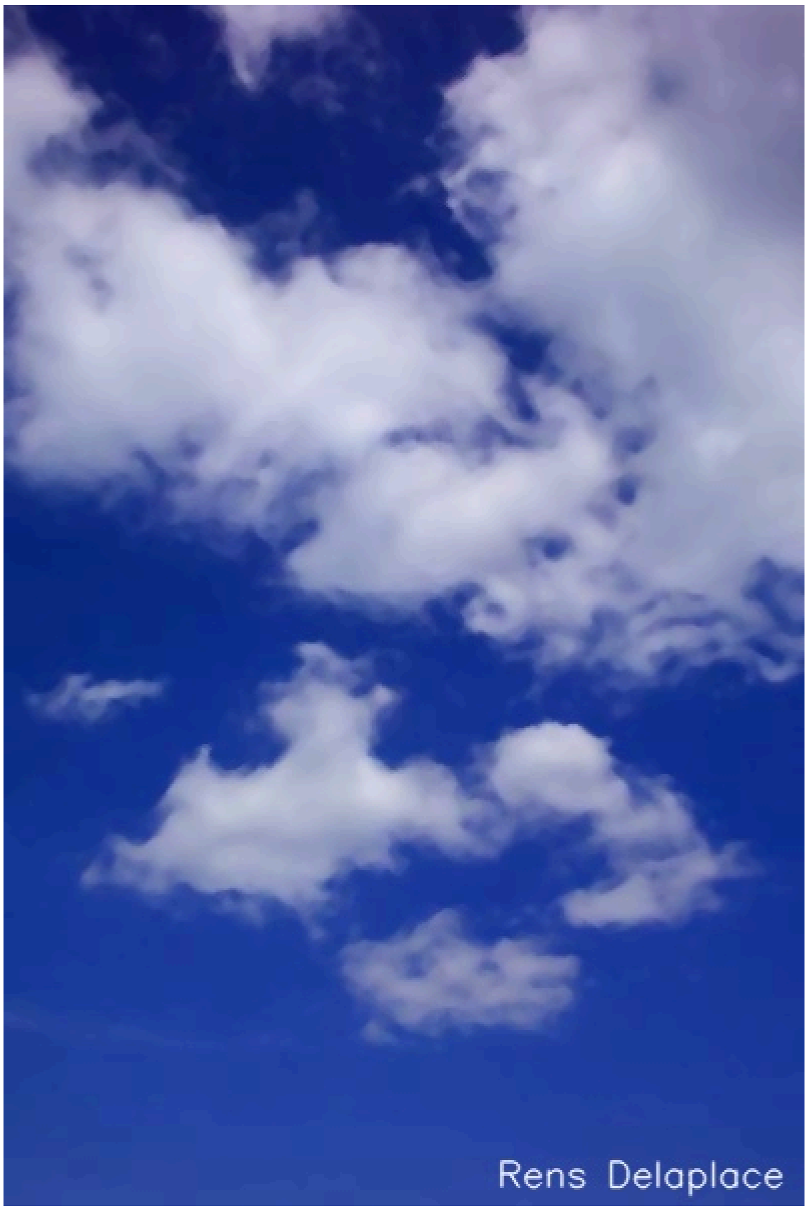


Assignment 11

> Apply median filtering on the same image.

In [13]:
```
saltpeppernoise_image = cv2.imread('img/saltandpeppernoise.png')
median_filtered_image = cv2.medianBlur(saltpeppernoise_image, 5)
median_filtered_image = add_name_to_image(median_filtered_image)
cv2.imwrite('out/assignement11.jpg', median_filtered_image)
```

```
Out[13]: True
```



Question 3

> Which result is preferable and why?

Median filtering is prefered for images with salt and pepper noise, as it discards outliers (salt & peper), while median filtering mixes these values with the rest of the image.

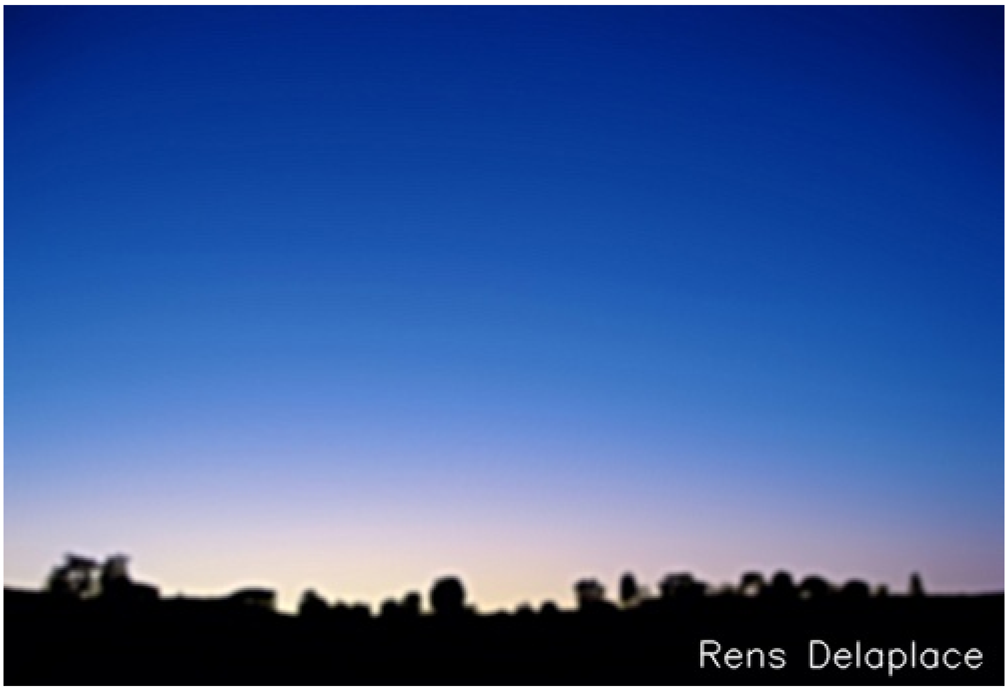
Exercise 6

Assignment 12

> Implement unsharp masking to sharpen unsharp.png. Make sure you do not get overflow in your datatype!

In [14]:
```
unsharp_image = np.float32(cv2.imread('img/unsharp.png'))

# Blur the image
unsharp_image_float = unsharp_image.astype(np.float32)
blurred_image = cv2.GaussianBlur(unsharp_image_float, (7, 7), 4)

# Subtract the blurred from the original
difference_image = unsharp_image_float - blurred_image

# Amplify the difference by multiplying it with a factor
amplified_difference = difference_image * 1.5

# Add this amplified difference image to the original image
sharpened_image = unsharp_image_float + amplified_difference

sharpened_image = np.clip(sharpened_image, 0, 255).astype(np.uint8)
sharpened_image = add_name_to_image(sharpened_image)
cv2.imwrite('out/assignement12.jpg', sharpened_image)
```

Out[14]:
```
True
```



Exercise 7

Assignment 13

> Write a program that blurs blots.png diagonally with the kernel below (mind the multiplication factor in front).

In [15]:
```
image = cv2.imread("img/blots.png")

kernel = (1/7) * np.eye(7, dtype=np.float32)
blurred_image = cv2.filter2D(image, -1, kernel)

blurred_image = add_name_to_image(blurred_image)
cv2.imwrite("out/assignement13.jpg", blurred_image)
```

```
Out[15]:  True
```