Manthan Gopal Dhole
201080020 IT Ty

Experiment No. 3
Subject: Digital Image Processing

Aim :

1. Write a program to generate an image of size entered by the user which exactly looks like
following. The size of the color boxes must be 1/8th the size of the original image.

2. Write THREE different functions which will create THREE images below based on options
(e.g. option 1, option 2, option 3) selected by the user.

Code them in such a way that system must ask user the following things (based on option)

1. Size of image(mXn)
2. Number of boxes
3. Number of horizontal and vertical lines
4. Size of the lines
5. Size of boxes
6. Color of boxes

Theory :

- OpenCV (Open Source Computer Vision) is a popular open-source library for computer vision and image processing tasks. It provides a wide range of functions and tools for working with images and videos. Among its features, OpenCV allows users to draw shapes on images and videos, including rectangles.

- To draw a rectangle on an image using OpenCV, you can use the rectangle function. This function takes as input the image where the rectangle will be drawn, the coordinates of the top-left and bottom-right corners of the rectangle, and the color and thickness of the rectangle's border.

Manthan Gopal Dhole **Experiment No. 3**
201080020 IT Ty **Subject: Digital Image Processing**

To draw a rectangle on the corner of an image, you need to know the size of the image and the size of the rectangle. Assuming that you want to draw the rectangle on the top-left corner of the image

- ***cv2.rectangle:*** This function is used to draw a rectangle on an image. It takes the following arguments:
  - img: the image on which to draw the rectangle
  - pt1: the top-left corner of the rectangle as a tuple of (x,y) coordinates
  - pt2: the bottom-right corner of the rectangle as a tuple of (x,y) coordinates
  - color: the color of the rectangle in BGR format
  - thickness: the thickness of the rectangle border in pixels. Use -1 to fill the rectangle with the color parameter.
- ***cv2.putText:*** This function is used to write text on an image. It takes the following arguments:
  - img: the image on which to write the text
  - text: the text to write on the image
  - org: the bottom-left corner of the text string as a tuple of (x,y) coordinates
  - font: the font type of the text (e.g., cv2.FONT_HERSHEY_SIMPLEX)
  - fontScale: the font scale factor that multiplies the font-specific base size
  - color: the color of the text in BGR format
  - thickness: the thickness of the text stroke in pixels
  - lineType: the type of the text stroke (e.g., cv2.LINE_AA)

These are just a few examples of the drawing functions available in OpenCV. There are many other functions for drawing shapes, filling regions, and applying special effects to images. By using these functions, you can create complex image processing applications in Python with OpenCV.

Code : Q1

```
from PIL import Image, ImageDraw
```

```
# Define the size of the main rectangle
width = 400
height = 300

# Create a new image with the specified size and white
background
image = Image.new("RGB", (width, height), "white")

# Define the size and position of the smaller rectangles
small_width = width // 8
small_height = height // 8
rect1_pos = (0, 0)
rect2_pos = (width - small_width, 0)
rect3_pos = (0, height - small_height)
rect4_pos = (width - small_width, height - small_height)
center_rect_pos = (width // 2 - small_width // 2, height // 2
- small_height // 2)

# Create a new drawing object for the image
draw = ImageDraw.Draw(image)

# Draw the smaller rectangles
draw.rectangle((rect1_pos, (rect1_pos[0] + small_width,
rect1_pos[1] + small_height)), fill="yellow")
draw.rectangle((rect2_pos, (rect2_pos[0] + small_width,
rect2_pos[1] + small_height)), fill="green")
draw.rectangle((rect3_pos, (rect3_pos[0] + small_width,
rect3_pos[1] + small_height)), fill="red")
draw.rectangle((rect4_pos, (rect4_pos[0] + small_width,
rect4_pos[1] + small_height)), fill="blue")
draw.rectangle((center_rect_pos, (center_rect_pos[0] +
small_width, center_rect_pos[1] + small_height)),
fill="#6699CC")

# Add the text "VJTI" to the center rectangle
# text_pos = (center_rect_pos[0] + small_width // 2,
center_rect_pos[1] + small_height // 2)
text_pos = (center_rect_pos[0] +small_width // 3,
center_rect_pos[1]+small_height // 3)
```

```
draw.text(text_pos, "VJTI", fill="white")

# Save the image as a file
image.save("1_rectangle_image.png")
```

Output:

Output : 

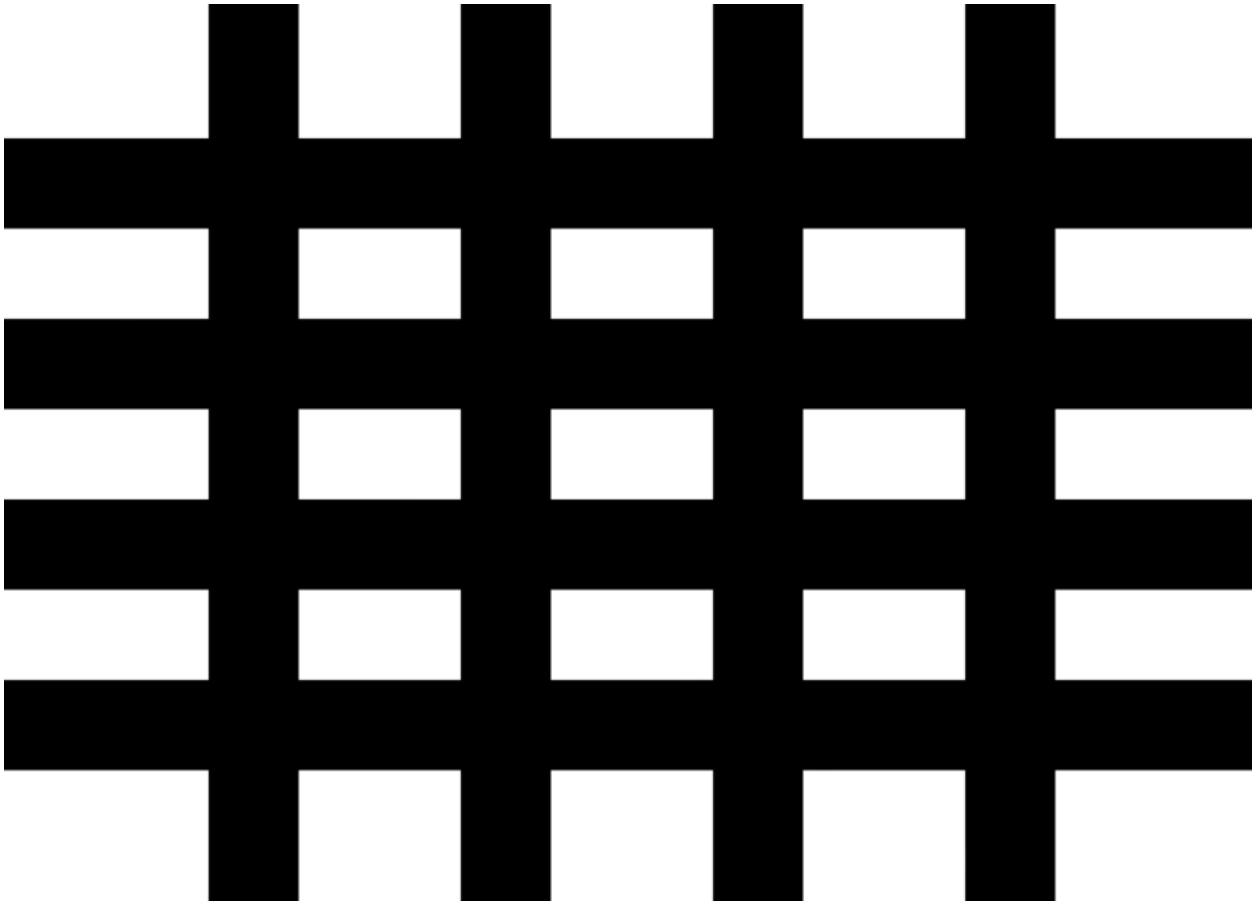
Code : Q2.1

```
# import required libraries
from PIL import Image, ImageDraw

# set the size of the grid and its borders
```

```
def FirstOption(width=700,height=500):
    # import required libraries

    # set the size of the image and the thickness of the grid lines
    width = 700
    height = 500
    line_thickness = height // 10

    # get the number of horizontal and vertical lines from the user
    num_horizontal_lines = int(input("Enter the number of horizontal
lines: "))
    num_vertical_lines = int(input("Enter the number of vertical lines:
"))

    # calculate the spacing between the lines
    horizontal_spacing = height // (num_horizontal_lines + 1)
    vertical_spacing = width // (num_vertical_lines + 1)

    # create a new image
    image = Image.new(mode='RGB', size=(width, height), color='white')

    # create a drawing object
    draw = ImageDraw.Draw(image)

    # draw the horizontal lines
    for i in range(num_horizontal_lines):
        if(i==num_horizontal_lines-1 and i==0):
            y = (i + 2) * horizontal_spacing
        else:
            y = (i + 1) * horizontal_spacing


        draw.line(xy=[(0, y), (width, y)], fill='black',
width=line_thickness)

    # draw the vertical lines
    for i in range(num_vertical_lines):
        x = (i + 1) * vertical_spacing
```

```
        draw.line(xy=[(x, 0), (x, height)], fill='black',
width=line_thickness)

    # save the image
    image.show()
    image.save('2_1.png')
```

Output :

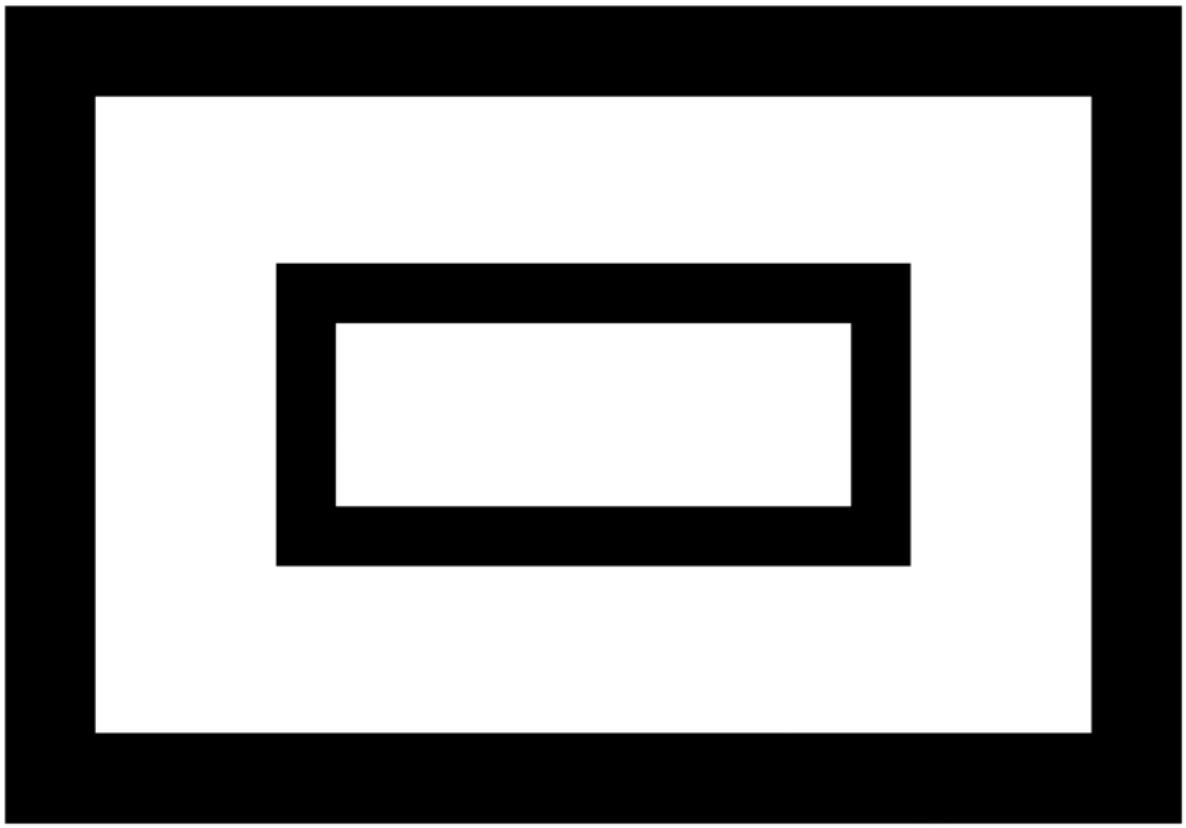


Code : Q2.2

```
#  Second Image
```

```
# import required libraries
from PIL import Image, ImageDraw

# set the size of the rectangle and its bordery
def SecondOption(width=700 ,height=500,color='black'):

    width = 700
    height = 500
    border_thickness = height // 10

    # set the size of the inner rectangle and its border
    inner_width = width // 2
    inner_height = height // 3
    inner_border_thickness = int(border_thickness/1.5)

    # calculate the position of the inner rectangle
    inner_x1 = (width - inner_width) //2
    inner_y1 = (height - inner_height) // 2
    inner_x2 = inner_x1 + inner_width
    inner_y2 = inner_y1 + inner_height

    # create a new image
    image = Image.new(mode='RGB', size=(width, height), color='white')

    # create a drawing object
    draw = ImageDraw.Draw(image)

    # draw the outer rectangle and its border
    outer_x1 = border_thickness // 2
    outer_y1 = border_thickness // 2
    outer_x2 = width - border_thickness // 2
    outer_y2 = height - border_thickness // 2
    draw.rectangle(xy=[(outer_x1, outer_y1), (outer_x2, outer_y2)],
outline=color, width=border_thickness)

    # draw the inner rectangle and its border
    draw.rectangle(xy=[(inner_x1, inner_y1), (inner_x2, inner_y2)],
outline=color, width=inner_border_thickness)
```

```
    # save the image
    image.show()
    image.save('2_2.png')

SecondOption(199,299,'red')
```



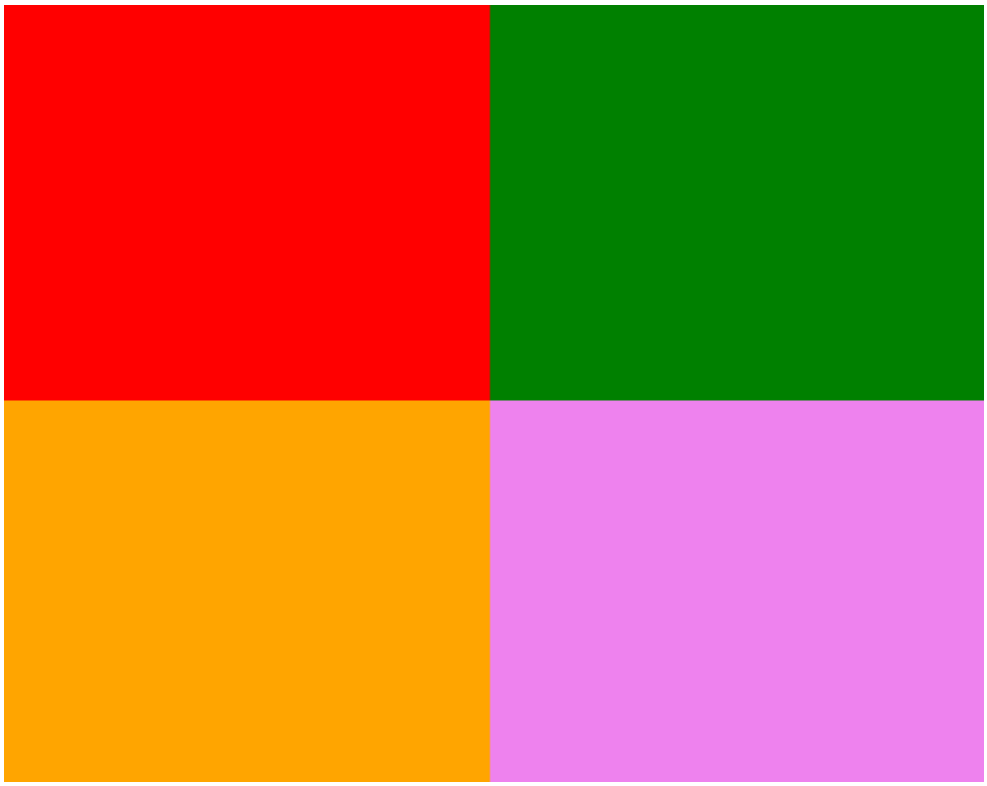
Code : Q2.3

```
from PIL import Image, ImageDraw

# Define the size of the main rectangle

def ThirdOption(width=400,height=300):
    # width = 400
    # height = 300
    colors=[]
```

```
    colors.append(input('Upper Left  :').lower())
    colors.append(input('Upper Right :').lower())
    colors.append(input('Lower Left  :').lower())
    colors.append(input('Lower Right :').lower())



    # Create a new image with the specified size and white background
    image = Image.new("RGB", (width, height), "white")

    # Define the size and position of the smaller rectangles
    small_width = width // 2
    small_height = height // 2
    rect1_pos = (0, 0)
    rect2_pos = (width - small_width, 0)
    rect3_pos = (0, height - small_height)
    rect4_pos = (width - small_width, height - small_height)
    center_rect_pos = (width // 2 - small_width // 2, height // 2 -
small_height // 2)

    # Create a new drawing object for the image
    draw = ImageDraw.Draw(image)

    # Draw the smaller rectangles
    try:
        draw.rectangle((rect1_pos, (rect1_pos[0] + small_width,
rect1_pos[1] + small_height)), fill=colors[0])
        draw.rectangle((rect2_pos, (rect2_pos[0] + small_width,
rect2_pos[1] + small_height)), fill=colors[1])
        draw.rectangle((rect3_pos, (rect3_pos[0] + small_width,
rect3_pos[1] + small_height)), fill=colors[2])
        draw.rectangle((rect4_pos, (rect4_pos[0] + small_width,
rect4_pos[1] + small_height)), fill=colors[3])
    except:
        print("Colors name are not specified corretly so, default colors
are choosen : ")
        draw.rectangle((rect1_pos, (rect1_pos[0] + small_width,
rect1_pos[1] + small_height)), fill="yellow")
```

```
        draw.rectangle((rect2_pos, (rect2_pos[0] + small_width,
rect2_pos[1] + small_height)), fill="green")
        draw.rectangle((rect3_pos, (rect3_pos[0] + small_width,
rect3_pos[1] + small_height)), fill="red")
        draw.rectangle((rect4_pos, (rect4_pos[0] + small_width,
rect4_pos[1] + small_height)), fill="blue")




    image.show()
    # Save the image as a file
    image.save("2_3.png")
ThirdOption(400,200)
```

Output :



```
option=int(input("Enter option :"))
if option==1:
    FirstOption()
elif option==2:
    SecondOption()
else:
    ThirdOption()
```

Conclusion :

OpenCV (Open Source Computer Vision) is a popular open-source library for computer vision and image processing tasks. It provides a wide range of functions and tools for working with images and videos. Among its features, OpenCV allows users to draw shapes on images and videos, including rectangles.