

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №2
по дисциплине «Информатика»
Тема: Введение в архитектуру компьютера

Студент гр. 2381

Зазуля И.А.

Преподаватель

Шевская Н.В.

Санкт-Петербург

2022

Цель работы.

Научиться обрабатывать изображение в Python с помощью библиотеки Pillow.

Задание.

Вариант 2.

Предстоит решить 3 подзадачи, используя библиотеку **Pillow (PIL)**.

Для реализации требуемых функций студент должен использовать **numpy** и **PIL**. Аргумент `image` в функциях подразумевает объект типа `<class 'PIL.Image.Image'>`

1) Рисование пентаграммы в круге

Необходимо написать функцию `pentagram()`, которая рисует на изображении пентаграмму в круге.

Функция `pentagram()` принимает на вход:

Изображение (`img`)

Координаты левого верхнего и правого нижнего угла квадрата, в который вписана окружность (`x0,y0,x1,y1`)

Толщину линий и окружности (`thickness`)

Цвет линий и окружности (`color`) - представляет собой список (`list`) из 3-х целых чисел

Функция должна вернуть обработанное изображение.

2) Инвертирование полос

Необходимо реализовать функцию `invert`, которая делит изображение на "полосы" и инвертирует цвет нечетных полос (счёт с нуля).

Функция `invert()` принимает на вход:

Изображение (`img`)

Ширину полос в пикселах (`N`)

Признак того, вертикальные или горизонтальные полосы (`vertical` - если True, то вертикальные)

Функция должна разделить изображение на вертикальные или горизонтальные полосы шириной N пикселей. И инвертировать цвет в нечетных полосах (счет с нуля). Последняя полоса может быть меньшей ширины, чем N .

3) Поменять местами 9 частей изображения

Необходимо реализовать функцию *mix*, которая делит квадратное изображение на 9 равных частей (**сторона изображения делится на 3**), и по правилам, записанным в словаре, меняет их местами.

Функция *mix()* принимает на вход:

Изображение (*img*)

Словарь с описанием того, какие части на какие менять (*rules*)

Функция должна вернуть обработанное изображение.

Выполнение работы.

Подключается библиотека *PIL*, из которой подключаются модули *Image*, *ImageDraw*. Также подключаются методы *sin*, *cos*, *pi* из модуля *math*.

Функция *pentagram()*.

Данная функция принимает на вход изображение (*img*), координаты левого верхнего и правого нижнего угла квадрата, в который вписана окружность (x_0, y_0, x_1, y_1), толщину линий и окружности (*thickness*), цвет линий и окружности (*color*).

Создаётся объект для рисования *drawing*. Затем с помощью метода *ImageDraw.ellipse* на объекте *drawing* рисуется окружность. Далее вычисляем радиус и координаты центра этой окружности.

Создаётся список *coords_pen* для хранения координат вершин пентаграммы. В который, путём использования цикла *for* заносим данные. После чего с помощью метода *ImageDraw.line* на объекте *drawing* рисуется линии соединяющие необходимые вершины пентаграммы.

Возвращается обработанное изображение.

Функция *invert()*.

Данная функция принимает на вход изображение(*img*), ширину полос в пикселях(*N*), признак того, вертикальные или горизонтальные полосы(*vertical* - если *True*, то вертикальные).

Путём обращения к полю *img.size* получаем размеры исходного изображения. Далее происходит проверка на то, какие полосы нам требуются. Циклом *for* проходим по каждой полосе и, если номер полосы нечётный, то проходим по каждому пикселю в этой полосе. Сначала находим исходный цвет пикселя с помощью метода *Image.getpixel()*, после чего заменяем его на противоположный и с помощью метода *Image.putpixel()* помещаем его на замену исходному.

Возвращается обработанное изображение.

Функция *mix()*.

Данная функция принимает на вход изображение (*img*), Словарь с описанием того, какие части на какие менять (*rules*).

Путём обращения к полю *img.size* получаем размеры исходного изображения. Находим размер стороны искомых квадратов. Создаётся список *coords* для хранения координат искомых квадратов по столбцам. Создаётся список *images* для хранения искомых изображений.

Далее находим координаты квадратов по столбцам и помещаем их в список *coords*. После упорядочиваем координаты и помещаем их в список *good_coords*. Следующим шагом с помощью метода *Image.crop()* обрезаем исходное изображение и получаем 9 одинаковых по размеру изображений, которые помещаем в список *images*. Финальным шагом с помощью метода *Image.paste()* по определённому правилу из словаря *rules* вставляем обрезанные изображения в исходное.

Возвращается обработанное изображение.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

Была освоена библиотека *Pillow*, посредством использования её на практике, для написании трёх функций: *pentagram()*, которая рисует пентаграмму в окружности, *invert()*, которая разбивает изображение на полосы противоположных цветов с определённой шириной, *mix()*, которая меняет вырезанные изображения из исходного местами.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
from PIL import Image, ImageDraw
from math import pi, cos, sin

def pentagram(img, x0, y0, x1, y1, thickness, color):
    color = tuple(color)
    drawing = ImageDraw.Draw(img, "RGB")
    drawing.ellipse((x0,y0,x1,y1),outline=color,width=thickness)
    r = int(abs(x0-x1)/2)
    center_x = int(x1 - abs(x0-x1)/2)
    center_y = int(y1 - abs(y0-y1)/2)
    coords_pen = []
    for i in range(0, 6):
        phi = (pi / 5) * (4 * i + 3 / 2)
        node_i = (int(center_x + r * cos(phi)),int(center_y +
r * sin(phi)))
        coords_pen.append(node_i)
    drawing.line(coords_pen, fill=color, width=thickness)
    return img

def invert(img, N,vertical):
    w,h = img.size
    if vertical:
        curr_width = 0
        count_l = w//N if w%N==0 else int(w/N)+1
        for i in range(count_l):
            if w%N!=0 and i==count_l-1:
                if i%2==1:
                    for j in range(curr_width, w):
                        for k in range(h):
                            pix = img.getpixel((j, k))
                            pix = tuple((255 - pix[i]) for i in
range(3))
                            img.putpixel((j, k), pix)
                else:
                    continue
            elif i%2==1:
                for j in range(curr_width, curr_width+N):
                    for k in range(h):
                        pix = img.getpixel((j,k))
                        pix = tuple((255 - pix[i]) for i in
range(3))
                        img.putpixel((j,k), pix)
                if curr_width + N <= w: curr_width += N
                else: break
    else:
        curr_height = 0
        count_l = h//N if h%N==0 else int(h/N)+1
        for i in range(count_l):
```

```

        if h%N!=0 and i==count_l-1:
            if i%2==1:
                for j in range(curr_height, h):
                    for k in range(w):
                        pix = img.getpixel((k, j))
                        pix = tuple((255 - pix[i]) for i in
range(3))
                            img.putpixel((k, j), pix)
                    else:
                        continue
            elif i%2==1:
                for j in range(curr_height, curr_height+N):
                    for k in range(w):
                        pix = img.getpixel((k, j))
                        pix = tuple((255 - pix[i]) for i in
range(3))
                            img.putpixel((k, j), pix)
                if curr_height + N <= h: curr_height += N
                else: break
    return img







def mix(img, rules):
    w, h = img.size
    side = int(h/3)
    images = []
    coords = []
    coords1 = [(0, int((side)*i)), (int(side)*1,
int((side)*(i+1))) for i in range(3)]
    coords2 = [(int(side), int((side)*i)), (int((side)*2),
int((side)*(i+1))) for i in range(3)]
    coords3 = [(int((side)*2), int((side)*i)), (int((side)*3),
int((side)*(i+1))) for i in range(3)]
    coords = [coords1] + [coords2] + [coords3]
    sort_coords = [coords[i][k] for k in range(3) for i in
range(3)]
    good_coords = []
    for i in range(9):
        good_coords.append([g for k in sort_coords[i] for g in
k])
    for l in range(9):
        images.append(img.crop((good_coords[l])))
    for t in range(9):
        img.paste(images[rules[t]], (good_coords[t][0],
good_coords[t][1]))
    return img

```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.2 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.			pentagram(img, 50,50,100,100,2,[0,0,0]) — верно.
2.			invert(img, 27, True) — верно.
3.			mix(img, {0:2,1:2,2:2,3:5,4:5,5:5,6:8,7:8,8:8}) — верно.