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. 3344 _____ Кузнецов Р.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Изучение обработки изображения на языке Python.

Задание.

Вариант 3

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 solve(), которая рисует на изображении пентаграмму в окружности.

Функция solve() принимает на вход:

- Изображение (img)
- координаты центра окружность (x,y)
- радиус окружности
- Толщину линий и окружности (thickness)
- Цвет линий и окружности (color) - представляет собой список (list) из 3-х целых чисел

Функция должна изменить исходное изображение и вернуть его изображение.

Примечание:

Вершины пентаграммы высчитывать по формуле:

phi = (pi/5)*(2*i+3/2)

node_i = (int(x0+r*cos(phi)),int(y0+r*sin(phi)))

x0,y0 - координаты центра окружности, в который вписана пентаграмма

r - радиус окружности

i - номер вершины от 0 до 4

2) Поменять местами участки изображения и поворот

Необходимо реализовать функцию solve, которая меняет местами два квадратных, одинаковых по размеру, участка изображений и поворачивает эти участки на 90 градусов по часовой стрелке, а затем поворачивает изображение на 90 градусов по часовой стрелке.

Функция solve() принимает на вход:

- Квадратное изображение (img)
- Координаты левого верхнего угла первого квадратного участка(x0,y0)
- Координаты левого верхнего угла второго квадратного участка(x1,y1)
- Длину стороны квадратных участков (width)

Функция должна сначала поменять местами переданные участки изображений. Затем повернуть каждый участок на 90 градусов по часовой стрелке. Затем повернуть всё изображение на 90 градусов по часовой стрелке.

Функция должна вернуть обработанное изображение, не изменяя исходное.

3) Средний цвет

Необходимо реализовать функцию solve, которая заменяет цвет каждого пикселя в области на средний цвет пикселей вокруг (не считая сам этот пиксель).

Функция solve() принимает на вход:

- Изображение (img)
- Координаты левого верхнего угла области (x0,y0)
- Координаты правого нижнего угла области (x1,y1)

Функция должна заменить цвета каждого пикселя в этой области на средний цвет пикселей вокруг.

Пиксели вокруг :

- 8 самых близких пикселей, если пиксель находится в центре изображения
- 5 самых близких пикселей, если пиксель находится у стенки
- 3 самых близких пикселя, если пиксель находится в угле

Функция должна вернуть обработанное изображение, не изменяя исходное.

Выполнение работы

Для работы были импортированы библиотеки *PIL*, *numpy, math, os*.

Вначале кода *OPENBLAS_NUM_THREADS* при помощи environ было установлено в значение 4 для корректной работы программы.

Была создана функция *swap(img, x0, y0, x1, y1, width)*, на вход которой подается изображение, координаты левого угла первого участка изображения и координаты левого угла второго участка изображения, а также длина стороны участков изображения. При помощи *copy* копируется изображение, метод *crop* вырезает заданные части изображения, *transpose* разворачивает вырезанные участки картинки на 270 градусов против часовой стрелки и *paste* вставляет обработанные части в изображение. Функция возвращает обработанное изображение, повернутое на 270 градусов против часовой стрелки.

Для выполнения следующей задачи были созданы 3 функции: главная *avg_color(img, x0, y0, x1, y1)* и побочные *near_pixels(img, x, y)* и *color_pixels(neighbors)*. Первая принимает на вход изображение, координаты левого угла и нижнего угла области. Копируется изображение и при помощи цикла *for* перебираются пиксели изображения. Пиксели вместе с исходным изображением отправляются в функцию *near_pixels*. В данной функции в переменные записывается минимальное и максимальное значение *x* и *y*. Причем если значения выходят за границы изображения, то в переменные все равно записываются границы изображения. Реализовано это при помощи *min* и *max*. Создается список *neighbors*, куда, затем при помощи цикла *for* добавляются соседние пиксели если они не являются текущим пикселем. Функция возвращает созданный список. Результат функции записывается в переменную *neighbors*. Затем эту переменную отправляет в другую функцию *color_pixels*. Там объявляются переменные *sum_r*, *sum_g*, *sum_b* и *count*. В *count* записывается длина списка *neighbors*. Цикл проходит по массиву и прибавляет значения *RGB* в переменные *sum_r*, *sum_g* и *sum_b*. После этого ищется среднее значение для каждого значения *RGB*, записывается в переменные *all_r, all_g, all_b* и функция возвращает их. Результат функции записывается в переменную *avg_color*, а потом

этот размещается на картинке. Главная функция возвращает обработанное изображение.

Была создана функция *pentagram(img, x, y, r, thickness, color)*, принимающая на вход изображение, координаты центра окружности, его радиус, толщина линий и их цвет. В переменную *color* записывается кортеж цветов. В переменную *drawing* записывается изображение на изменение. Рисуется на изображении окружность и затем создается список для вершин звезды. При помощи цикла вычисляются координаты вершин и добавляются их в список. Потом вершины соединяются линиями и функция возвращает полученное изображение.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	swap(Image.open("krab1.jpeg"), 0, 0, 150, 150, 150)	image	-
2.	avg_color(Image.open("pixels.jpeg", 0, 0, 500, 500)	result_img	-
3.	pentagram(Image.new("RGB", (300, 300)), 62, 72, 136, 135, 4, [112, 11, 205])	img	-

Выводы

Во время выполнения лабораторной работы была изучена обработка изображений на Python. Были получены знания для работы с библиотекой *Pillow*.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb_2.py

```
import os
os.environ["OPENBLAS_NUM_THREADS"] = "4"

from PIL import Image, ImageDraw
import numpy as np
import math

def swap(img, x0, y0, x1, y1, width):
    image = img.copy()

    first = image.crop((x0, y0, x0 + width, y0 + width))
    second = image.crop((x1, y1, x1 + width, y1 + width))

    first = first.transpose(Image.Transpose.ROTATE_270)
    second = second.transpose(Image.Transpose.ROTATE_270)

    image.paste(first, (x1, y1))
    image.paste(second, (x0, y0))

    return image.transpose(Image.Transpose.ROTATE_270)

def near_pixels(img, x, y):
    x_min = max(0, x-1)
    x_max = min(img.width-1, x+1)
    y_min = max(0, y-1)
    y_max = min(img.height-1, y+1)

    neighbors = []

    for i in range(x_min, x_max + 1):
        for j in range(y_min, y_max +1):
            if (i, j) != (x, y):
                neighbors.append(img.getpixel((i, j)))

    return neighbors

def color_pixels(neighbors):
    sum_r, sum_g, sum_b = 0, 0, 0
    count = len(neighbors)

    for color in neighbors:
        r, g, b = color
        sum_r += r
        sum_g += g
        sum_b += b

    all_r = int(sum_r / count)
    all_g = int(sum_g / count)
    all_b = int(sum_b / count)
```

```
    return (all_r, all_g, all_b)

def avg_color(img, x0, y0, x1, y1):
    result_img = img.copy()

    for i in range(x0, x1 + 1):
        for j in range(y0, y1 + 1):
            neighbors = near_pixels(img, i, j)
            avg_color = color_pixels(neighbors)
            result_img.putpixel((i, j), avg_color)

    return result_img

def pentagram(img, x, y, r, thickness, color):
    color = tuple(color)
    drawing = ImageDraw.Draw(img)
    drawing.ellipse(((x-r,y-r), (x+r,y+r)), width = thickness, outline =
color,)
    versh = []

    for i in range(5):
        phi = (math.pi/5)*(2*i+3/2)
        node_i = (int(x+r*math.cos(phi)),int(y+r*math.sin(phi)))
        versh.append(node_i)

    drawing.line((versh[0], versh[2]), fill = tuple(color), width =
thickness)
    drawing.line((versh[1], versh[3]), fill = tuple(color), width =
thickness)
    drawing.line((versh[2], versh[4]), fill = tuple(color), width =
thickness)
    drawing.line((versh[3], versh[0]), fill = tuple(color), width =
thickness)
    drawing.line((versh[4], versh[1]), fill = tuple(color), width =
thickness)
    return img
```