

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. 3341

Преподаватель

Шаповаленко

Е.В.

Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Решить 3 подзадачи, используя библиотеку *Pillow (PIL)*. Для реализации требуемых функций необходимо использовать *numpy* и *PIL*. Аргумент *image* в функциях подразумевает объект типа `<class 'PIL.Image.Image'>`

Задание

Вариант 3

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 окружности (*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:

$$\phi_i = (\pi / 5) * (2 * i + 3 / 2)$$

$$\text{node_}i = (\text{int}(x_0 + r * \cos(\phi_i)), \text{int}(y_0 + r * \sin(\phi_i)))$$

x_0, y_0 - 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(x_0, y_0)
- Координаты левого верхнего угла второго квадратного участка(x_1, y_1)
- 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.

Пример входной картинке (300x300) и переданных параметров:



$x_0 = 0$; $y_0 = 0$; $x_1 = 150$; $y_1 = 150$; $width = 150$;

Изображение после первого этапа (замена участков и поворот этих участков на 90 градусов):



Итоговое изображение:



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.

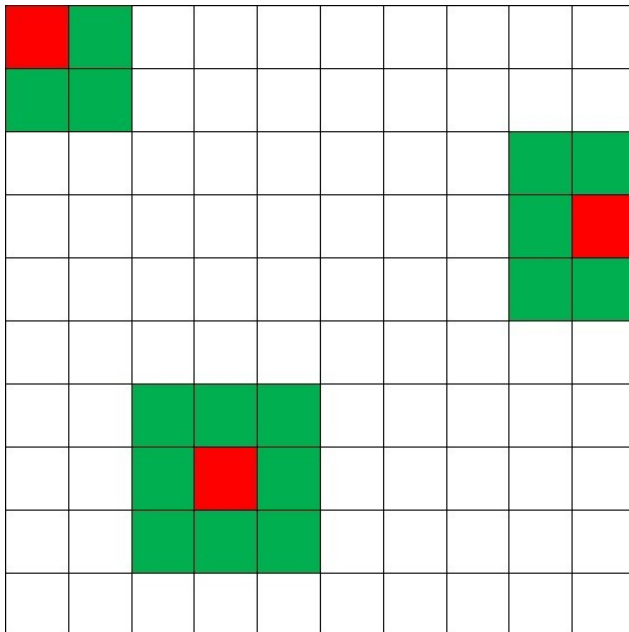
Пиксели вокруг:

- 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.

Средний цвет - берется целая часть от среднего каждой компоненты из *rgb*. ($\text{int}(\text{sum}(r) / \text{count})$, $\text{int}(\text{sum}(g) / \text{count})$, $\text{int}(\text{sum}(b) / \text{count})$)

Пример получения среднего цвета:



Для красных пикселей, берется средний цвет зеленых пикселей вокруг него.

Можно реализовывать дополнительные функции.

Выполнение работы

Импортируются библиотеки *numpy* и *PIL*, необходимые для работы функций.

Функция *swar* принимает в качестве аргументов изображение *img*, координаты левого верхнего угла первого фрагмента (x_0, y_0), координаты левого верхнего угла второго фрагмента (x_1, y_1), а также размер квадратных областей *width*.

Сначала функция вырезает два фрагмента изображения из областей, определенных координатами. Происходит вращение каждого фрагмента на 270 градусов против часовой стрелки (что аналогично повороту на 90 градусов по часовой). Затем создается копия исходного изображения *result*. Первый фрагмент вставляется на место второго фрагмента, а второй — на место первого. В конце *result* поворачивается на 270 градусов против часовой стрелки и возвращается в качестве результата работы функции.

Функция *avg_color* принимает в качестве аргументов изображение *img*, координаты левого верхнего и правого нижнего углов области, в которой необходимо заменить цвет пикселей (x_0, y_0 и x_1, y_1 соответственно).

Функция создает копию изображения и заменяет все пиксели в необходимой области в соответствии с результатом работы функции *get_avg_color*. После чего *result* возвращается в качестве результата работы функции.

Функция *get_avg_color* принимает в качестве аргументов изображение *img* и координаты пикселя (x и y).

Функция создает список *neighbors*, в который сохраняет цвета пикселей-соседей в соответствии с координатами пикселя, в котором заменяется цвет, и размерами изображения.

После чего высчитывается среднее значение цвета по каждой из составляющей (красный, зеленый и синий) и возвращается в качестве результата работы функции в виде кортежа.

Функция *pentagram* принимает в качестве аргументов изображение *img*, координаты центра окружности (*x* и *y*), ее радиус *r*, толщину и цвет линий (*thickness* и *color* соответственно).

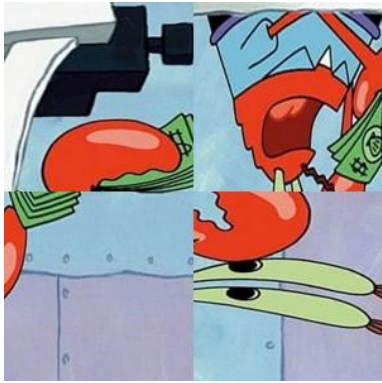
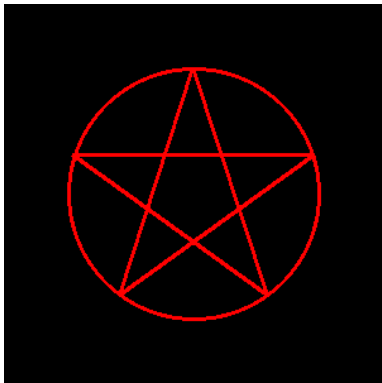
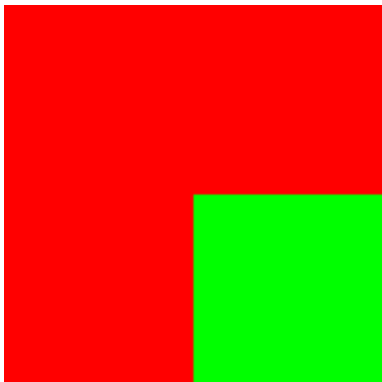
Функция преобразует список *color* в кортеж. После этого создается объект класса *PIL.ImageDraw.Draw* для рисования на изображении. В соответствии с переданными в функцию параметрами рисуется окружность. Далее высчитываются координаты вершин пентаграммы, и в соответствии с ними и другими параметрами рисуется пентаграмма. В качестве результата работы функции возвращается измененное исходное изображение.

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	<code>swap(Image.open("krab_1.jpg"), 0, 0, 150, 150, 150)</code>		-
2.	<code>pentagram(Image.new("RGB", (300, 300), 0), 150, 150, 100, 3, [255, 0, 0])</code>		-
3.	<code>img=Image.new("RGB", (500, 500), "red") img.paste(Image.new("RGB", (250, 250), "lime"), (250, 250)) avg_color(img, 200, 200, 499, 499)</code>		На границе красного и зеленого цветов имеется темная полоса – среднее значение двух цветов

Выводы

В ходе выполнения работы были успешно решены три подзадачи, использующие библиотеку *Pillow (PIL)*. Для разработки необходимых функций, в которых аргумент *img* представляет собой объект класса *<PIL.Image.Image>*, были применены модули *numpy* и *PIL*.

Были изучены основы работы с этими библиотеками, полученные знания применены на практике.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

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

def swap(img, x0, y0, x1, y1, width):
    first = img.crop((x0, y0, x0 + width, y0 + width))
    first = first.rotate(270)

    second = img.crop((x1, y1, x1 + width, y1 + width))
    second = second.rotate(270)

    result = img.copy()
    result.paste(first, (x1, y1))
    result.paste(second, (x0, y0))
    result = result.rotate(270)

    return result

def get_avg_color(img, x, y):
    width, height = img.size

    neighbors = []
    for i in range(-1, 2):
        for j in range(-1, 2):
            if (0 <= (x + i) < width) and (0 <= (y + j) < height):
                neighbors.append(img.getpixel((x + i, y + j)))
    neighbors.remove(img.getpixel((x, y)))

    resultColor = [0, 0, 0]
    for color in neighbors:
        for i in range(3):
            resultColor[i] += color[i]

    for i in range(3):
        resultColor[i] = int(resultColor[i] / len(neighbors))
    return tuple(resultColor)

def avg_color(img, x0, y0, x1, y1):
    result = img.copy()
    for x in range(x0, x1+1):
        for y in range(y0, y1+1):
            result.putpixel((x, y), get_avg_color(img, x, y))
    return result

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)), fill = None,
outline = color, width = thickness)
```

```

nodes = [(0, 0)] * 5
for i in range(5):
    phi = (np.pi / 5) * (2 * i + 3 / 2)
    nodes[i] = (int(x + r * np.cos(phi)), int(y + r *
np.sin(phi)))

    drawing.line((nodes[0], nodes[2], nodes[4], nodes[1], nodes[3],
nodes[0]), fill = color, width = thickness)

return img

```