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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Информатика»

Тема: Введение в архитектуру компьютера

Студент гр. 3341 Гребенюк В.А.

Преподаватель Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Целью работы является освоение работы с библиотекой Pillow языка Python на примере использующей их программы.

Задание

Вариант 4

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 отрезка. Отрезок определяется:

- координатами начала
- координатами конца
- цветом
- толщиной.

Необходимо реализовать функцию *user_func()*, рисующую на картинке отрезок

Функция *user_func()* принимает на вход:

- изображение;
- координаты начала (x0, y0);
- координаты конца (x1, y1);
- цвет;
- толщину.

Функция должна вернуть обработанное изображение.

2) Преобразовать в Ч/Б изображение (любым простым способом).

Функционал определяется:

- Координатами левого верхнего угла области;
- Координатами правого нижнего угла области;
- Алгоритмом, если реализовано несколько алгоритмов преобразования изображения (по желанию студента).

Нужно реализовать 2 функции:

- *check_coords(image, x0, y0, x1, y1)* - проверяет координаты области (x0, y0, x1, y1) на корректность (они должны быть неотрицательными, не превышать размеров изображения, поскольку x0, y0 - координаты левого верхнего угла, x1, y1 - координаты правого нижнего угла, то x1 должен быть больше x0, а y1 должен быть больше y0);
- *set_black_white(image, x0, y0, x1, y1)* - преобразовывает заданную область изображения в черно-белый (используйте для конвертации параметр '1'). В этой функции должна вызываться функция проверки, и, если область некорректна, то должно быть возвращено исходное изображение без изменений. *Примечание:* поскольку черно-белый формат изображения (greyscale) является самостоятельным форматом, а не вариацией RGB-формата, для его получения необходимо использовать метод *Image.convert*.

3) Найти самый большой прямоугольник заданного цвета и перекрасить его в другой цвет. Функционал определяется:

- Цветом, прямоугольник которого надо найти
- Цветом, в который надо его перекрасить.

Написать функцию *find_rect_and_recolor(image, old_color, new_color)*, принимающую на вход изображение и кортежи rgb-компонент старого и нового цветов. Она выполняет задачу и возвращает изображение. При необходимости можно писать дополнительные функции.

Выполнение работы

Функции: •. • set_black_white(image: Image, x0, y0, x1, y1): преобразует область изображения в черно-белую. • find_rect_and_recolor(image: Image, old_color: tuple, new_color: tuple): находит прямоугольник с заданным цветом и перекрашивает его в новый цвет.

Импортированные модули: • PIL (Image, ImageDraw) • numpy (np)

В данном коде используются следующие методы и функции модулей PIL и NumPy: • ImageDraw.Draw(image).line((x0, y0, x1, y1), fill, width): рисует линию на изображении с заданными координатами, цветом и толщиной. • image.crop((x0, y0, x1, y1)): обрезает изображение по заданным координатам. • _img.convert("1"): преобразует изображение в черно-белое. • image.paste(_img, (x0, y0)): вставляет изображение в заданные координаты. • image.convert("RGB"): преобразует изображение в RGB. • np.zeros((width,), dtype=int): создает массив нулей заданной длины. • np.all(np.array(image) == old_color, axis=2): создает битовую маску, где True соответствует пикселям старого цвета. • ImageDraw.Draw(image).rectangle(max_pos, new_color): рисует прямоугольник на изображении с заданными координатами и цветом.

Функции:

- *user_func*(*image: Image, x0, y0, x1, y1, fill, width*): рисует линию на изображении с заданными координатами, цветом и толщиной.
- *check_coords(image: Image, x0, y0, x1, y1)*: проверяет, находятся ли координаты в пределах изображения
- *set_black_white(image: Image, x0, y0, x1, y1)*: преобразует область изображения в черно-белую.
- *find_rect_and_recolor(image: Image, old_color: tuple, new_color: tuple)*: находит прямоугольник с заданным цветом и перекрашивает его в новый цвет.

Импортированные модули:

- *numpy (np)*
- *PIL (Pillow)*

В данном коде используются следующие методы и функции модулей PIL и NumPy:

- *ImageDraw.Draw(image).line((x0, y0, x1, y1), fill, width)*: рисует линию на изображении с заданными координатами, цветом и толщиной.
- *image.crop((x0, y0, x1, y1))*: обрезает изображение по заданным координатам.
- *_img.convert("1")*: преобразует изображение в черно-белое.
- *image.paste(_img, (x0, y0))*: вставляет изображение в заданные координаты.
- *image.convert("RGB")*: преобразует изображение в RGB.
- *np.zeros((width,), dtype=int)*: создает массив нулей заданной длины.
- *np.all(np.array(image) == old_color, axis=2)*: создает битовую маску, где True соответствует пикселям старого цвета.
- *ImageDraw.Draw(image).rectangle(max_pos, new_color)*: рисует прямоугольник на изображении с заданными координатами и цветом.

Выводы

Библиотека Pillow языка Python была успешно усвоена. Эта библиотека обладает большим количеством функций для работы с изображениями**.**

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

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


# Задача 1
def user_func(image: Image, x0, y0, x1, y1, fill, width):
    ImageDraw.Draw(image).line((x0, y0, x1, y1), fill, width)
    return image


# Задача 2
def check_coords(image: Image, x0, y0, x1, y1):
    if x0 > x1 or y0 > y1:
        return False
    if x0 < 0 or y0 < 0:
        return False
    if x1 > image.width or y1 > image.height:
        return False
    return True


def set_black_white(image: Image, x0, y0, x1, y1):
    if not check_coords(image, x0, y0, x1, y1):
        return image
    _img = image.crop((x0, y0, x1, y1))
    _img = _img.convert("1")
    image.paste(_img, (x0, y0))
    return image


# Задача 3
def find_rect_and_recolor(image: Image, old_color: tuple,
new_color: tuple):
    image = image.convert("RGB")
    width, height = image.size
    max_pos = (0, 0, 0, 0)
    max_area = 0
    bars = np.zeros((width,), dtype=int)

    bit_mask = np.all(np.array(image) == old_color, axis=2)

    # processing top to bottom (cus why not)
    for y in range(height):
        left_boundary = np.zeros(width, dtype=int)
        right_boundary = np.zeros(width, dtype=int)

        # y is bottom line of bars
```

```
                    bars[bit_mask[y]] += 1    # numpy shenanigans :D
(vectorised)
                bars[~bit_mask[y]] = 0
                # this is same as:
                # for x in range(width):
                #     if bit_mask[y][x]:
                #         bars[x] += 1
                #     else:
                #         bars[x] = 0

                temp = []
                for x in range(width):
                    while temp and bars[temp[-1]] >= bars[x]:
                        temp.pop()
                    left_boundary[x] = temp[-1] if temp else -1
                    temp.append(x)

                temp = []
                for x in range(width - 1, -1, -1):
                    while temp and bars[temp[-1]] >= bars[x]:
                        temp.pop()
                    right_boundary[x] = temp[-1] if temp else width
                    temp.append(x)

                # area for every bar
                for x in range(width):
                                area = bars[x] * (right_boundary[x] -
left_boundary[x] - 1)
                    if area > max_area:
                        max_area = area
                        max_pos = (
                            left_boundary[x] + 1,
                            y - bars[x] + 1,
                            right_boundary[x] - 1,
                            y,
                        )

            if max_area != 0:
                ImageDraw.Draw(image).rectangle(max_pos, new_color)
            return image


        # if __name__ == "__main__":
        #     with Image.open("test.png") as im:
        #         o = find_rect_and_recolor(im, (0, 0, 0), (200, 0, 0))
        #     o.save("o.png")
```