

**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(ка) гр. 3383

Логинова А.Ю.

Преподаватель

Иванов Д. В

Санкт-Петербург

2023

### **Цель работы.**

Написать программу, которая решает 3 подзадачи, используя библиотеку Pillow(PIL). Для реализации требуемых функций студент необходимо использовать **numpy** и **PIL**.

Аргумент `image` в функциях подразумевает объект типа `<class 'PIL.Image.Image'>`.

### **Задание.**

#### **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.

Задание на лабораторную работу в исходной формулировке. Указывается полностью. Указывается вариант задания, если есть. Обратите внимание, что

форматирование текста задания должно быть таким, как и основной текст в отчете (вставляйте текст задания «без сохранения исходного форматирования», или с «применением форматирования документа»).

### **Выполнение работы.**

Для выполнения 1 задачи был использован метод Draw, находящийся в модуле ImageDraw. С помощью этого метода можно нарисовать фигуры, в данном случае было необходимо нарисовать линию. В метод передаются координаты конца и начала, возвращается обновленное изображение.

Для выполнения 2 задачи были использованы методы Crop и Convert, находящиеся в модуле Image. Перед тем как сделать изображение черно-белым, необходимо проверить координаты, это осуществляется посредством сравнения точек. Далее с помощью значения '1' картинка *image* становится черно-белой.

Для выполнения задачи 3 был использован методы из numpy np.zeros(), который создает нулевую матрицу ширины переданной на вход картинки. В этой матрице будет храниться максимальная высота столбцов окрашенных пикселей в картинке по соответствующим координатам (под “окрашенными пикселями” подразумеваются те пиксели, цвет которых совпадает с переданным цветом на вход), на момент прохождения строки, соответствующая этапу обработки. Далее создаются начальные переменные *area\_size*, *left\_i*, *left\_j*, *right\_i*, *right\_j*. Далее матрица итеративно заполняется значениями.

Далее создается переменная *heights\_set*, которая хранит в себе набор неповторяющихся элементов матрицы *vector*. Создаются начальные переменные: *max\_area*, *max\_area\_left\_i*, *max\_area\_right\_i*, *max\_area\_left\_j*, *max\_area\_right\_j*. И запускается цикл, пробегающий по значениям *heights\_set*. Цикл ищет самый большой набор рядом стоящих столбцов, удовлетворяющих требованию быть не меньше соответствующего числа в этом множестве. Цикл проходит по всем числам в множестве. Самый большой ряд стоящих столбцов позволяет найти самую большую площадь для этого числа. Переменная *chunk* хранит в себе количество рядом стоящих столбцов. Под столбцами понимается

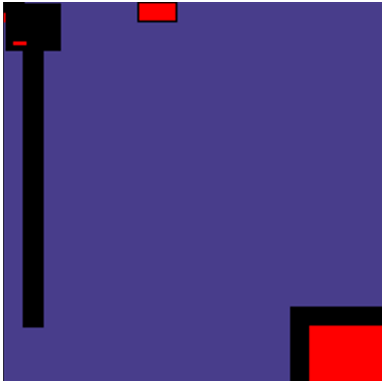
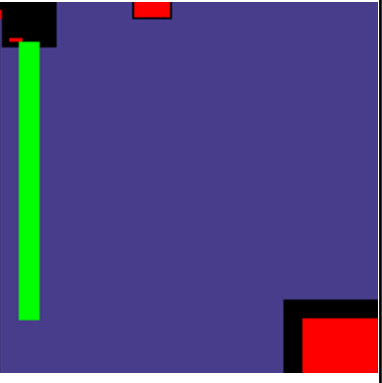
столбец подряд идущих окрашенных пикселей. После того как будет найдена самая большая площадь для числа, она сохраняется в переменную, которая позже сравнивается между всеми числами, таким образом находится самая большая площадь до n-й строки. Таким образом, пройдя по всем n строкам, найдется самая большая площадь прямоугольника.

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.			Наибольший прямоугольник цвета был закрашен зеленым.

### Выводы.

Был исследован модуль PIL, который позволяет создавать, открывать, обрабатывать картинки. Также был изучен ImageDraw, позволяющий рисовать на любой картинке.

Была решена задача, которая преобразовывает картинку в черно-белую, которая рисует на картинке линию, и которая находит самый большой прямоугольник заданного цвета и закрашивает его цветом, заданным пользователем. Задачи были решены с помощью ранее изученных модулей.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

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

def user_func(image, x0, y0, x1, y1, fill, width):
    draw = ImageDraw.Draw(image)
    draw.line((x0, y0, x1, y1), fill, width)
    return image

def check_coords(image, x0, y0, x1, y1):
    width, height = image.size

    coord_not_negative = x0 >= 0 and x1 >= 0 and y0 >= 0 and y1 >=
0
    coord_fit = x1 <= width and y1 <= height
    x1y1_greater_x0y0 = x1 > x0 and y1 > y0

    if coord_not_negative and coord_fit and x1y1_greater_x0y0:
        return True
    return False

def set_black_white(image, x0, y0, x1, y1):
    coord_check_not_passed = not (check_coords(image, x0, y0, x1,
y1))

    if coord_check_not_passed:
        return image

    img_crp = image.crop((x0, y0, x1, y1)).convert('1')
    image.paste(img_crp, (x0, y0, x1, y1))
    return image
```

```

def find_rect(image, color):
    pixels = image.load()
    width, height = image.size

    vector = np.zeros(width, dtype=int)

    area_size, left_i, left_j, right_i, right_j = 0, 0, 0, 0, 0

    for j in range(height):
        for i in range(width):
            if pixels[i, j] == color:
                vector[i] += 1
            else:
                vector[i] = 0

        heights_set = set(vector)
        heights_set.remove(0)

        max_area, max_area_left_i, max_area_right_i,
max_area_left_j, max_area_right_j = 0, 0, 0, 0, 0
        for set_elem in heights_set:
            max_chunk, max_chunk_i, max_chunk_j, chunk, chunk_i,
chunk_j = 0, 0, 0, 0, 0, 0
            for vec_i in range(width):
                if vector[vec_i] >= set_elem:
                    if chunk == 0:
                        chunk_i = vec_i
                        chunk_j = set_elem
                        chunk += 1
                    else:
                        if chunk > max_chunk:
                            max_chunk = chunk
                            max_chunk_i = chunk_i
                            max_chunk_j = chunk_j
                        chunk = 0
            else:
                if chunk > max_chunk:

```

```

        max_chunk = chunk
        max_chunk_i = chunk_i
        max_chunk_j = chunk_j

    new_area_size = max_chunk * set_elem
    if new_area_size > max_area:
        max_area = new_area_size
        max_area_left_i = max_chunk_i
        max_area_right_i = max_area_left_i + max_chunk - 1
        max_area_left_j = j - set_elem + 1
        max_area_right_j = j

    if max_area > area_size:
        area_size = max_area
        left_i = max_area_left_i
        right_i = max_area_right_i
        left_j = max_area_left_j
        right_j = max_area_right_j

    return area_size, (left_i, left_j), (right_i, right_j)

def recolor_rect(image, i, j, new_color):
    drawing = ImageDraw.Draw(image)
    drawing.rectangle((i, j), new_color)
    return image

def find_rect_and_recolor(image, old_color, new_color):
    area_size, left, right = find_rect(image, old_color)
    if area_size > 0:
        upd_image = recolor_rect(image, left, right, new_color)
        return upd_image
    else:
        return image

```