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

ОТЧЕТ

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

Студент гр. 3384 _____ Рудаков А.Л

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2023

Цель работы.

Изучить основы работы с библиотекой Pillow. Применить навыки на практике, написав код, для решения поставленных задач.

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.

В начале программы была подключена библиотека *PIL*, из которой были импортированы классы *Image* и *ImageDraw*.

Для решения задачи #1 была написана функция *def user_func(image, x0, y0, x1, y1, fill, width),* которая принимает в аргументах изображение *image*,

координаты начала отрезка *x0* и *y0*, координаты конца отрезка *x1* и *y1*, цвет отрезка *fill* и толщина отрезка *width*. Внутри нее был инициализирован *drawing* (специальный объект), при помощи метода *ImageDraw.Draw(image)*. Благодаря ему можно «рисовать» на изображении. Далее при помоще метода *.line()* была нарисована прямая, имеющая координаты начала *(x0,y0)*, координаты конца *(x1,y1)*, цвет прямой *fill* и толщину прямой *width*. Изображение возвращается из функции при помощи оператора *return*.

Далее была написана первая функция, требующаяся для решения задачи #2. *def check_coords(image, x0, y0, x1, y1)* - функция проверки правильности координат, которая принимает в качестве аргументов изображение *image*, координаты левого верхнего угла *x0* и *y0*, координаты правого нижнего угла *x1* и *y1*. Внутри нее инициализирована переменная *count* отвечающая за хранение информации о количестве несовпадений, которой изначально присвоено значение 0. Далее идет проверка на то, что все координаты положительны, если это не так, то к значению *count* прибавляется 1. Далее инициализируются переменные *width* и *height*, значение которым присваивается из метода *image.size*, возвращающего размеры изображения. После этого идет проверка на то, что полученные координаты не превосходят размеры изображения, если это не так, то значение *count* увеличивается на 1. После этого идет проверка на правильность последовательности координат (должно быть *x0<=x1* и *y0<=y1*), если это не так, то значение *count* увеличивается на 1. после этого, если значение *count* не равно 0, то возвращается *False*, в противном случае возвращается *True*.

Далее была написана вторая функция, требующаяся для решения задачи #2. *def set_black_white(image, x0, y0, x1, y1)* — функция преобразования части изображения в ч/б, оторая принимает в качестве аргументов изображение *image*, координаты левого верхнего угла *x0* и *y0*, координаты правого нижнего угла *x1* и *y1*. Внутри нее идет проверка, если значение функции *check_coords* это *True*, то выполняются действия внутри условного оператора. В строке *polygon=image.crop((x0,y0,x1,y1))* при помощи метода *.crop()* вырезается

полигон из изображения *image* с координатами верхнего левого угла *(x0,y0)* и нижнего правого угла *(x1,y1)*. Далее при помощи метода *.convert('1')* вырезанный полигон конвертируется в ч/б. Далее в изображение *image* при помощи метода *.paste()* вставляется полученный полигон, верхний левый угол вставки в координатах *(x0,y0)*. Изображение возвращается из функции при помощи оператора *return*. изображение *image*, координаты левого верхнего угла *x0* и *y0*, координаты правого нижнего угла *x1* и *y1*. Внутри нее идет проверка, если значение функции *check_coords* это *True*, то выполняются действия внутри условного оператора. В строке *polygon=image.crop((x0,y0,x1,y1))* при помощи метода *.crop()* вырезается полигон из изображения *image* с координатами верхнего левого угла *(x0,y0)* и нижнего правого угла *(x1,y1)*. Изображение возвращается из функции при помощи оператора *return*.

Далее была написана *def find_rect_and_recolor(image, old_color, new_color)* — функция для поиска наибольшего прямоугольника определенного цвета и для замены цвета этого прямоугольника на другой цвет. В аргументах функции *image* — изображение, *old_color* — цвет наибольшего прямоугольника, который нужно найти, *new_color* — цвет, на который нужно заменить цвет найденного прямоугольника. Внутри нее были инициализированы 2 кортежа *old_color_new = tuple(old_color) и new_color_new = tuple(new_color)*, который при помощи функции *tuple()* изменяют тип *list* на *tuple*. Также было иницализированы *max_volume* — отвечающая за хранение максимальной площади и переменные *width* и *height,* значение которым присваивается из метода *image.size,* возвращающего размеры изображения. Далее при помощи метода *.load()* изображение было разбито на матрицу пикселей *pixels*. Инициализированы переменные *coor_x0, coor_x1, coor_y0, coor_y1*, отвечающие за хранение координат наибольшего прямоугольника. Далее идут вложенные циклы, благодаря которым выполняется проход по всей матрице пикселей. Внутри условный оператор, проверяющий то, является ли цвет данного пикселя исконным цветом. Если это так, то инициализируются *x0_new, x1_new,* которым присваивается значение *x* и *y0_new, y1_new*, которым присваивается значение *y*.

Далее идет цикл *while pixels[x1_new,y1_new]==old_color_new,* выполняющийся до момента, пока цвет нового пикселя не станет другим. Внутри к значению *y1_new* прибавляется 1, проверяется на то, что *y1_new* не равно высоте изображения, если же это не так, то выполняется выход из цикла при помощи оператора *break*. Далее значение *y1_new* уменьшается на 1. Далее идет цикл *while pixels[x1_new,y1_new]==old_color_new*, выполняющийся до момента, пока цвет нового пикселя не станет другим. Внутри к значению *x1_new* прибавляется 1, проверяется на то, что *x1_new* не равно ширине изображения, если же это не так, то выполняется выход из цикла при помощи оператора *break*. Далее значение *x1_new* уменьшается на 1. Далее находится площадь прямоугольника *volume=(x1_new-x0_new+1)*(y1_new-y0_new+1)* и идет проверка, если площадь текущего прямоугольника больше площади наибольшего найденного прямоугольника, то выполняются действия внутри условного оператора. Значению *max_volume* присваивается значение *volume*, значению *coor_x0 — x0_new*, значению *coor_x1 — x1_new*, значению *coor_y0 — y0_new*, значению *coor_y1 — y1_new*. Следующими идут вложенные циклы, проходящие по найденному наибольшему прямоугольнику и заменяют цвета пикселей на новый цвет *new_color_new.* Изображение возвращается из функции при помощи оператора *return*.

Выводы.

Был написан код, решающий поставленные задачи. В функциях использовались такие методы модуля pillow как: *Image.size, ImageDraw.line(), Image.load(), ImageDraw.Draw(), Image.paste(), Image.crop(), Image.convert()*.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main_lb2.py

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

# Задача 2
def check_coords(image, x0, y0, x1, y1):
    count=0
    if x0<0 or x1<0 or y0<0 or y1<0:
        count+=1
    width,height=image.size
    if x0>width or x1>width or y0>height or y1>height:
        count+=1
    if x0>x1 or y0>y1:
        count+=1
    if count==0:
        return True
    else:
        return False

def set_black_white(image, x0, y0, x1, y1):
    if check_coords(image,x0,y0,x1,y1):
        polygon=image.crop((x0,y0,x1,y1))
        polygon_new=polygon.convert('1')
        image.paste(polygon_new,(x0,y0))
    return image

# Задача 3
def find_rect_and_recolor(image, old_color, new_color):
    old_color_new = tuple(old_color)
    new_color_new = tuple(new_color)
    max_volume=0
    width, height = image.size
    pixels = image.load()
    coor_x0 = coor_x1 = coor_y0 = coor_y1 =0
    for x in range(width):
        for y in range(height):
            if pixels[x,y]==old_color_new:
                x0_new = x1_new = x
                y0_new = y1_new = y
                while pixels[x1_new,y1_new]==old_color_new:
                    y1_new+=1
                    if y1_new==height:
                        break
                y1_new -= 1
                while pixels[x1_new,y1_new]==old_color_new:
                    x1_new+=1
                    if x1_new==width:
                        break
                x1_new -= 1
                volume=(x1_new-x0_new+1)*(y1_new-y0_new+1)
```

```
                if volume>max_volume:
                    max_volume=volume
                    coor_x0=x0_new
                    coor_x1=x1_new
                    coor_y0=y0_new
                    coor_y1=y1_new
for y in range(coor_y0, coor_y1 + 1):
    for x in range(coor_x0, coor_x1 + 1):
        pixels[x, y] = new_color_new
return image
```