**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 |  | Романов А.К. |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2022

## Цель работы

Целью лабораторной работы является изучение модуля Pillow языка программирования Python. Для этого требуется решить три подзадачи с использованием библиотек Pillow и numpy.

Для достижения поставленной цели требуется решить следующие

задачи:

1) Ознакомиться с модулем Pillow.

2) Научиться его использовать.

3) Необходимо разработать функции, которые работают с объектами типа *<class ‘PIL.Image.Image’>*.

## Задание

Вариант работы №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

Для решения задачи было реализовано несколько функций для каждой из задач, указанных в условии.

* *user\_func(image, x0, y0, x1, y1, fill, width)*: Эта функция использует библиотеку PIL для создания экземпляра *ImageDraw* и рисует линию между точками (*x0, y0*) и (*x1, y1*) указанной ширины(*width*) и цвета(*fill*).
* *check\_coords(image, x0, y0, x1, y1)*: Проверяет, лежат ли координаты (*x0, y0*) и (*x1, y1*) в пределах размеров изображения *image* и образуют ли они прямоугольник с правильными координатами (то есть x1 > x0 и y1 > y0).
* *set\_black\_white(image, x0, y0, x1, y1)*: Если координаты валидны (проверяется через *check\_coords*), функция вырезает часть изображения (*crop*), преобразует ее в черно-белое (двоичное) изображение и вставляет обратно на изначальное место.
* *find\_rect\_and\_recolor(image, old\_color, new\_color)*: Эта функция находит наибольший прямоугольник, состоящий из пикселей с цветом *old\_color* в изображении *image*. Затем она перекрашивает этот прямоугольник в цвет *new\_color*. Данная функция работатет в связке сфункцией *filler*: изначально создается матрица отображения пикселей изображения (где 1 соответствуют пикселям *old\_color*, а все остальные элементы равны 0). Далее данная матрица обрабатывается при помощи функции *filler*, после чего по найденным координатам перекрашивается самый большой прямоугольник.
* *filler(x, y, wd, hg, matrix)*: Функция заполняет область, начиная с точки (*x, y*), в матрице matrix размера *wd x hg* (матрица отображения пикселей — см. выше), помечая все смежные пиксели, которые имеют значение 1 в *matrix*. Возвращает координаты ограничивающего прямоугольника этой области.

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 |
|  |  |  | Рисовние линии |
|  |  |  | Переведение части изображения в ч/б |
|  |  |  | Перекрашивание самого большого прямоугольника заданного цвета |

## Выводы

Были разработаны функции, которые работают с объектами типа *<class ‘PIL.Image.Image’>*, выполняющие соответствующие задачи, используя библиотек Pillow и numpy.

В результате выполнения лабораторной работы были изучены и использованы на практике функции библиотек Pillow и numpy.

# Приложение А Исходный код программы

Название файла: main.py

Программный код:

import numpy as np

import PIL

import numpy as np

from PIL import Image, ImageDraw

def user\_func(image, x0, y0, x1, y1, fill, width):

art = ImageDraw.Draw(image)

art.line((x0, y0, x1, y1), fill, width)

return image

def check\_coords(image, x0, y0, x1, y1):

coords = [x0, x1, y0, y1]

size = image.size

if all(x>=0 for x in coords):

if all(x<=size[0] for x in coords[0:2]) and all(x<=size[1] for x in coords[3:]):

if x1>x0 and y1>y0:

return True

return False

def set\_black\_white(image, x0, y0, x1, y1):

if check\_coords(image, x0, y0, x1, y1):

edit = image.crop((x0,y0,x1,y1))

edit = edit.convert("1")

image.paste(edit, (x0,y0))

return image

def find\_rect\_and\_recolor (image, old\_color, new\_color):

pix\_data = image.load()

size = image.size

wd, hg = size[0], size[1]

matrix = np.zeros((wd, hg))

for x in range (wd):

for y in range(hg):

if image.getpixel((x, y)) == old\_color:

matrix[x, y] = 1

max\_rect\_coordinates = (0, 0, 0, 0)

max\_rect\_size = 0

for x in range (wd):

for y in range(hg):

if matrix[x, y]== 1:

rect\_coordinates = filler(x, y, wd, hg, matrix)

rect\_size = (rect\_coordinates[2] - rect\_coordinates[0]) \* (rect\_coordinates[3] - rect\_coordinates[1])

if rect\_size > max\_rect\_size:

max\_rect\_size = rect\_size

max\_rect\_coordinates = rect\_coordinates

for x in range(max\_rect\_coordinates[0], max\_rect\_coordinates[2]):

for y in range(max\_rect\_coordinates[1], max\_rect\_coordinates[3]):

pix\_data[x, y] = new\_color

return image

def filler (x ,y, wd, hg, matrix):

stack = [(x, y)]

coord\_mn = [wd, hg]

coord\_mx = [0, 0]

while stack:

x\_cur, y\_cur = stack.pop()

if 0 <= x\_cur < wd and 0 <= y\_cur < hg and matrix[x\_cur, y\_cur] == 1:

matrix[x\_cur, y\_cur] = 2

coord\_mn = [min(coord\_mn[0], x\_cur), min(coord\_mn[1], y\_cur)]

coord\_mx = [max(coord\_mx[0], x\_cur), max(coord\_mx[1], y\_cur)]

stack.append((x\_cur + 1, y\_cur))

stack.append((x\_cur - 1, y\_cur))

stack.append((x\_cur, y\_cur + 1))

stack.append((x\_cur, y\_cur - 1))

return (coord\_mn[0], coord\_mn[1], coord\_mx[0]+1, coord\_mx[1]+1)