МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Рязанский государственный радиотехнический университет

имени В.Ф. Уткина»

Кафедра ВПМ

**Лабораторная работа 2. Вычисления на основе взаимодействия сервисов**

Выполнил ст. гр. 3413М

Зайцев В.О.

Проверил

Князьков П.А.

Рязань, 2023

Код тетради сервера xmlrpc\_server.ipynb:

from xmlrpc.server import SimpleXMLRPCServer

from xmlrpc.server import SimpleXMLRPCRequestHandler

import xmlrpc.client

import datetime

import pandas as pd

import pickle

import numpy as np

class RequestHandler(SimpleXMLRPCRequestHandler):

    rpc\_paths = ('/RPC2',)

server = SimpleXMLRPCServer(("localhost", 8080),

                            requestHandler=RequestHandler, allow\_none=True)

log\_server = xmlrpc.client.ServerProxy("http://localhost:8018")

# Добавление в лог через сервер

def add\_log(log\_line):

    try:

        log\_server.add\_log(log\_line)

    except ConnectionRefusedError:

        pass

# Тест

def ping():

    add\_log('ping')

    return True

server.register\_function(ping, 'ping')

# Время сервера

def now():

    add\_log('now')

    return datetime.datetime.now()

server.register\_function(now, 'now')

# Отображение строкового вида, типа и значений

def show\_type(arg):

    add\_log('type')

    return (str(arg), str(type(arg)), arg)

server.register\_function(show\_type, 'type')

# Сумма

def test\_sum(a, b):

    add\_log('sum')

    return a + b

server.register\_function(test\_sum, 'sum')

# Степень

def test\_pow(a, b):

    add\_log('pow')

    return a\*\*b

server.register\_function(test\_pow, 'pow')

def in\_black\_list(candidate):

    add\_log('in\_black\_list')

    frame = pd.read\_csv('bad\_boys2.csv', header=0, sep=',', encoding='utf8')

    is\_in\_black\_list = any(frame['Surname'] == candidate)

    boy = predicate\_to\_string(is\_in\_black\_list)

    return candidate + ": "+ boy

def predicate\_to\_string(predicate):

    if (predicate == True):

        return "bad\_boy"

    else:

        return "good\_boy"

server.register\_function(in\_black\_list, 'in\_black\_list')

def to\_binary(data\_arg):

    data = data\_arg.data

    return xmlrpc.client.Binary(data)

server.register\_function(to\_binary, 'to\_binary')

def inversion(data\_arg):

    add\_log('inversion')

    picture\_matrix = pickle.loads(data\_arg.data)

    height = picture\_matrix.shape[0]

    width = picture\_matrix.shape[1]

    colors = len(picture\_matrix.shape)

    if (colors == 3):

        for i in range(height):

            for j in range(width):

                picture\_matrix[i][j][0] = 255 - picture\_matrix[i][j][0]

                picture\_matrix[i][j][1] = 255 - picture\_matrix[i][j][1]

                picture\_matrix[i][j][2] = 255 - picture\_matrix[i][j][2]

    elif(colors == 2):

        for i in range(height):

            for j in range(width):

                picture\_matrix[i][j] = 255 - picture\_matrix[i][j]

    result = pickle.dumps(picture\_matrix)

    return xmlrpc.client.Binary(result)

server.register\_function(inversion, 'inversion')

def mirror(data\_arg):

    add\_log('mirror')

    picture\_matrix = pickle.loads(data\_arg.data)

    height = picture\_matrix.shape[0]

    width = picture\_matrix.shape[1] - 1

    for i in range(height):

        for j in range(width // 2):

            original = picture\_matrix[i][j].copy()

            picture\_matrix[i][j] = picture\_matrix[i][width - j].copy()

            picture\_matrix[i][width - j] = original

    result = pickle.dumps(picture\_matrix)

    return xmlrpc.client.Binary(result)

server.register\_function(mirror, 'mirror')

def picture\_binarisation(data\_arg, threshold):

    add\_log('picture\_binarisation')

    picture\_matrix = pickle.loads(data\_arg.data)

    height = picture\_matrix.shape[0]

    width = picture\_matrix.shape[1]

    colors = len(picture\_matrix.shape)

    for i in range(height):

        for j in range(width):

            if(colors == 3):

                if(picture\_matrix[i][j][0] > threshold and picture\_matrix[i][j][1] > threshold and picture\_matrix[i][j][2] > threshold):

                    picture\_matrix[i][j][0] = 255

                    picture\_matrix[i][j][1] = 255

                    picture\_matrix[i][j][2] = 255

                else:

                    picture\_matrix[i][j][0] = 0

                    picture\_matrix[i][j][1] = 0

                    picture\_matrix[i][j][2] = 0

            elif(colors == 2):

                if(picture\_matrix[i][j] > threshold):

                    picture\_matrix[i][j] = 255

                else:

                    picture\_matrix[i][j] = 0

    result = pickle.dumps(picture\_matrix)

    return xmlrpc.client.Binary(result)

server.register\_function(picture\_binarisation, 'picture\_binarisation')

def in\_black\_list\_by(surname, name, patronym, birth):

    add\_log('in\_black\_list\_by')

    frame = pd.read\_csv('bad\_boys2.csv', header=0, sep=',', encoding='utf8')

    is\_in\_black\_list = any((frame['Surname'] == surname) & (frame['Name'] == name) & (frame['Patronym'] == patronym) & (frame['Birth'] == birth))

    boy = predicate\_to\_string(is\_in\_black\_list)

    return surname + ": " + boy

server.register\_function(in\_black\_list\_by, 'in\_black\_list\_by')

print ("Listening on port 8080...")

server.serve\_forever()

Код тетради клиента xmlrpc\_client.ipynb:

In [1]:

*# xmlrpc\_client.ipynb*

# xmlrpc\_client.ipynb

import xmlrpc.client

import pickle

import pandas as pd

server = xmlrpc.client.ServerProxy("http://localhost:8080")

stats\_server = xmlrpc.client.ServerProxy("http://localhost:8018")

print ('Ping', server.ping())

print ('Server datetime', server.now())

print ('View, type, value:', server.type(2))

print ('View, type, value:', server.type(2.))

print ('View, type, value:', server.type('My string'))

print ('View, type, value:', server.type("My string"))

print ('View, type, value:', server.type([1,2,3]))

print ('View, type, value:', server.type(["one", "two", "three"]))

print( 'Sum 2 + 3 :', server.sum(2, 3))

print( 'Sum 2 ^ 3 :', server.pow(2, 3))

Out[1]:

Ping True

Server datetime 20231023T19:04:48

View, type, value: ['2', "<class 'int'>", 2]

View, type, value: ['2.0', "<class 'float'>", 2.0]

View, type, value: ['My string', "<class 'str'>", 'My string']

View, type, value: ['My string', "<class 'str'>", 'My string']

View, type, value: ['[1, 2, 3]', "<class 'list'>", [1, 2, 3]]

View, type, value: ["['one', 'two', 'three']", "<class 'list'>", ['one', 'two', 'three']]

Sum 2 + 3 : 5

Sum 2 ^ 3 : 8

In [2]:

*#отображение монохромного изображения*

from PIL import Image # Работа с изображением

from pylab import \* # Отображение

import xmlrpc.client

# Режим отображения внутри ноутбука

%matplotlib inline

# img = Image.open('11.bmp')

img = Image.open('Jellyfish.jpg')

img\_arr = array(img)

# Исходное изображение

gray()

imshow(img\_arr)

Out[2]:

<matplotlib.image.AxesImage at 0x1de5a56dfd0>



In [3]:

# Тест бинарной передачи данных

#pimg = img\_arr.dumps()

pimg = pickle.dumps(img\_arr) # универсально

img\_bin = xmlrpc.client.Binary(pimg)

#img\_bin2 = server.to\_binary(img\_bin)

img\_bin2 = server.to\_binary(img\_bin)

#img\_arr2 = np.loads(img\_bin2.data)

img\_arr2 = pickle.loads(img\_bin2.data) # универсально

# Изображение после возрата с сервера

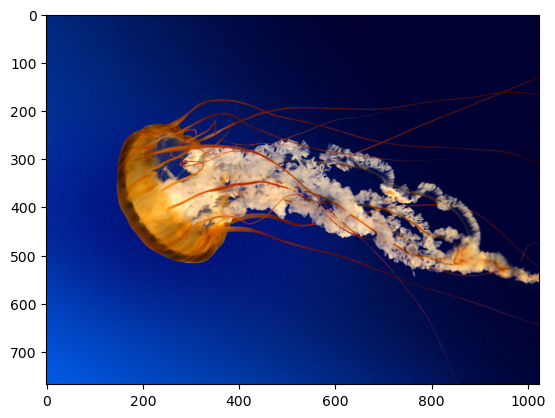
imshow(img\_arr2)

*# Исходное изображение*

imshow(img\_arr)

Out[3]:

<matplotlib.image.AxesImage at 0x1de5a7d9a90>



In [4]:

# Инверсия цвета изображения через сервер

def inv\_color(img\_arr\_in):

    pimg = pickle.dumps(img\_arr\_in)

    img\_bin = xmlrpc.client.Binary(pimg)

    #img\_bin2 = server.inversion(img\_bin)

    img\_bin2 = server.inversion(img\_bin)

    img\_arr\_out = pickle.loads(img\_bin2.data)

    return img\_arr\_out

img\_arr\_inv = inv\_color(img\_arr)

# Изображение с инверсией цвета

imshow(img\_arr\_inv)

Out[4]:

<matplotlib.image.AxesImage at 0x1de5a880c50>



#Зеркальное отображение

pimg = pickle.dumps(img\_arr)

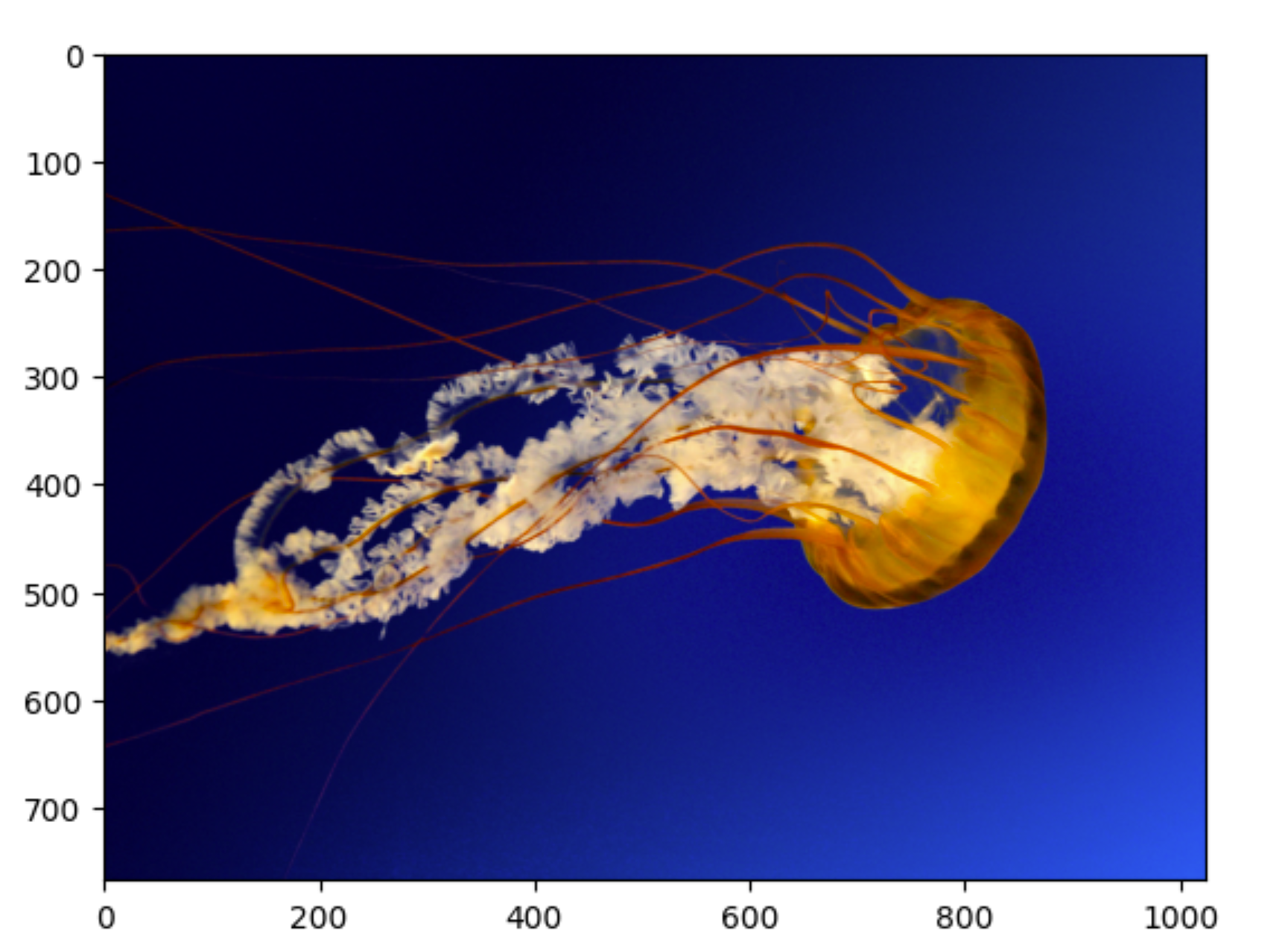
img\_bin = xmlrpc.client.Binary(pimg)

#mirr\_img\_bin = server.mirror(img\_bin)

mirr\_img\_bin = server.mirror(img\_bin)

img\_arr = pickle.loads(mirr\_img\_bin.data)

imshow(img\_arr)



#Бинаризация изображения

pimg = pickle.dumps(img\_arr)

img\_bin = xmlrpc.client.Binary(pimg)

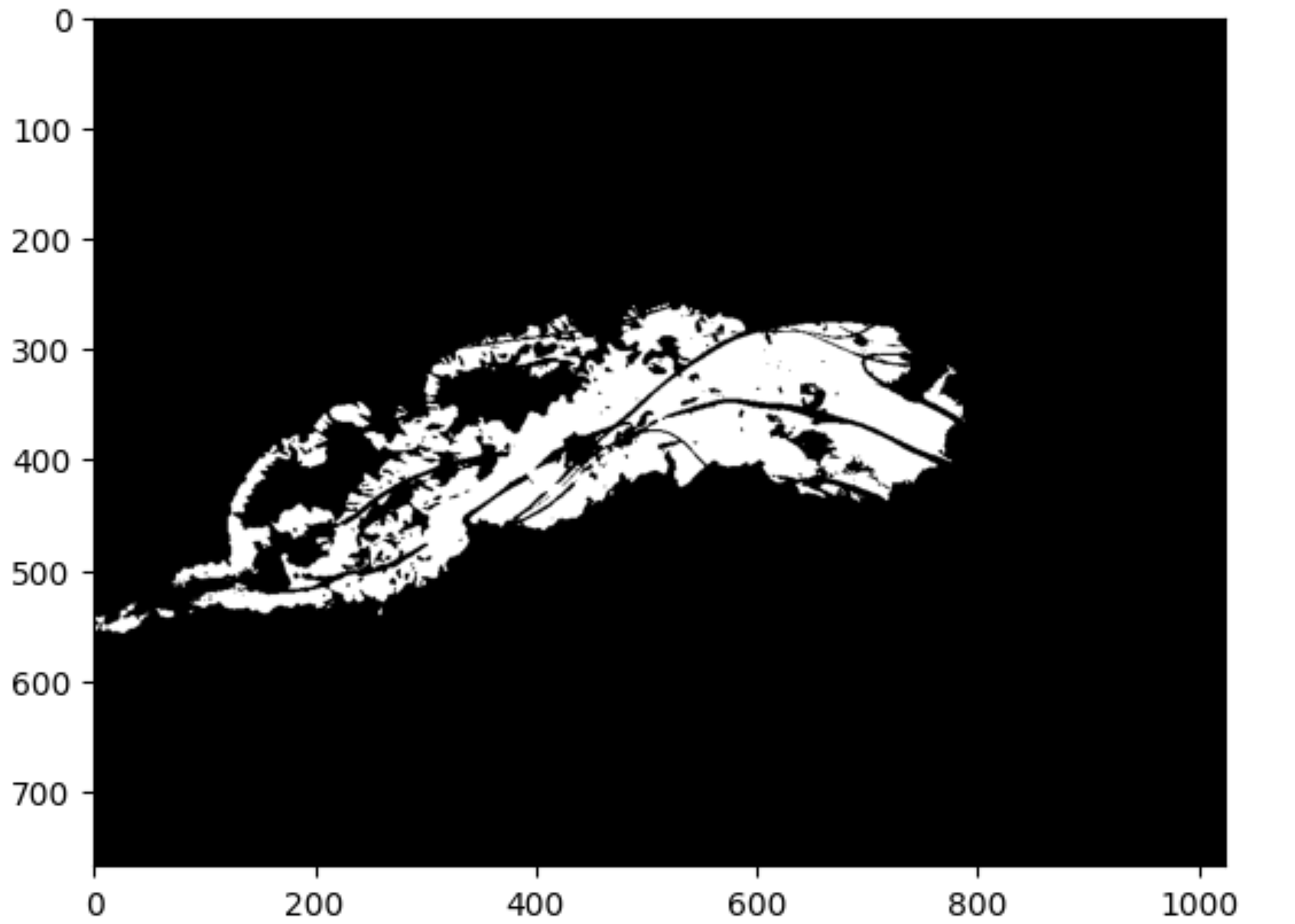
threshold = 100

#binary\_img\_bin = server.picture\_binarisation(img\_bin, porog)

binary\_img\_bin = server.picture\_binarisation(img\_bin, threshold)

binary\_img\_bin = pickle.loads(binary\_img\_bin.data)

imshow(binary\_img\_bin)



print(stats\_server.logs())

[['2023-10-10 18:01:57', 'ping'], ['2023-10-10 18:02:01', 'now'], ['2023-10-10 18:02:05', 'type'], ['2023-10-10 18:02:09', 'type'], ['2023-10-10 18:02:13', 'type'], ['2023-10-10 18:02:17', 'type'], ['2023-10-10 18:02:21', 'type'], ['2023-10-10 18:02:25', 'type'], ['2023-10-10 18:02:29', 'sum'], ['2023-10-10 18:02:34', 'pow'], ['2023-10-21 19:00:40', 'ping'], ['2023-10-21 19:00:44', 'now'], ['2023-10-21 19:00:49', 'type'], ['2023-10-21 19:00:53', 'type'], ['2023-10-21 19:00:57', 'type'], ['2023-10-21 19:01:01', 'type'], ['2023-10-21 19:01:05', 'type'], ['2023-10-21 19:01:09', 'type'], ['2023-10-21 19:01:13', 'sum'], ['2023-10-21 19:01:17', 'pow'], ['2023-10-23 19:04:44', 'ping'], ['2023-10-23 19:04:48', 'now'], ['2023-10-23 19:04:52', 'type'], ['2023-10-23 19:04:56', 'type'], ['2023-10-23 19:05:00', 'type'], ['2023-10-23 19:05:04', 'type'], ['2023-10-23 19:05:08', 'type'], ['2023-10-23 19:05:12', 'type'], ['2023-10-23 19:05:17', 'sum'], ['2023-10-23 19:05:21', 'pow'], ['2023-10-23 19:06:50', 'inversion'], ['2023-10-23 19:07:04', 'in\_black\_list'], ['2023-10-23 19:07:18', 'mirror'], ['2023-10-23 19:07:27', 'picture\_binarisation'], ['2023-10-23 19:07:41', 'in\_black\_list\_by']]

print(stats\_server.logs\_slice\_by\_operation('type'))

[['2023-10-10 18:02:05', 'type'], ['2023-10-10 18:02:09', 'type'], ['2023-10-10 18:02:13', 'type'], ['2023-10-10 18:02:17', 'type'], ['2023-10-10 18:02:21', 'type'], ['2023-10-10 18:02:25', 'type'], ['2023-10-21 19:00:49', 'type'], ['2023-10-21 19:00:53', 'type'], ['2023-10-21 19:00:57', 'type'], ['2023-10-21 19:01:01', 'type'], ['2023-10-21 19:01:05', 'type'], ['2023-10-21 19:01:09', 'type'], ['2023-10-23 19:04:52', 'type'], ['2023-10-23 19:04:56', 'type'], ['2023-10-23 19:05:00', 'type'], ['2023-10-23 19:05:04', 'type'], ['2023-10-23 19:05:08', 'type'], ['2023-10-23 19:05:12', 'type']]

print(stats\_server.logs\_slice\_by\_time('2023-10-10 18:02:09', '2023-10-10 18:02:21'))

[['2023-10-10 18:02:09', 'type'], ['2023-10-10 18:02:13', 'type'], ['2023-10-10 18:02:17', 'type'], ['2023-10-10 18:02:21', 'type']]

print(stats\_server.logs\_slice\_by\_time2('2023-10-10 18:02:09', '2023-10-20 18:02:21', 'type'))

[['2023-10-10 18:02:05', 'type'], ['2023-10-10 18:02:09', 'type'], ['2023-10-10 18:02:13', 'type'], ['2023-10-10 18:02:17', 'type']]

По результатам данной работы были получены следующие навыки:

1. Работа с сервером: приобретение опыта в разработке и модификации функций сервера.

2. Обработка изображений: приобретение опыта в модификации функции для работы с монохромными изображениями, добавление функции бинаризации и разворота изображения.

3. Работа без использования библиотечных функций: приобретение навыков в реализации функционала без использования сторонних библиотек для инверсии, бинаризации и разворота изображений.

4. Приобрели навыки разработки сфокусированных на одной задаче сервисов.

В данной работе было продемонстрировано, что использование сервис-ориентированной архитектуры вычислений является целесообразным.

Сервис-ориентированная архитектура позволяет эффективно организовать работу с различными функциями сервера, а также упростить их модификацию и расширение.