МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

**ОТЧЕТ**

по лабораторной работе № 6
на тему: «Сетевое программирование»
по дисциплине: «Программирование на языке Python»
Вариант № 7

Выполнила: Карпикова С.П. Шифр: 170580
Институт приборостроения, автоматизации и информационных технологий
Направление: 09.03.04 «Программная инженерия»
Группа: 71-ПГ
Проверили: Захарова О.В., Раков В.И.

Отметка о зачете:

Дата: «____» _____ 2019 г.

Орел, 2019 г.

## Задание:

Разработать клиент-серверное приложение для вычисления выражений.

Требования к клиенту:

- отправка на сервер введенного пользователем выражения (числа в восьмеричном представлении, знаки «+» и «-»), например: «33+7-10+2»;

- получение результата вычисления выражения;

- удобный графический интерфейс.

Требования к серверу:

- вычисление полученного от клиента выражения и отправка результата клиенту.

## Решение:

```
import sys
from gui import ClientAppGUI
from network import ClientAppNetwork


class ClientApp:

    def __init__(self, port, server_ip):
        print("Инициализация клиента, порт: {} , IP: {}".format(port, server_ip))
        self.network = ClientAppNetwork(port, server_ip)
        self.GUI = ClientAppGUI(self.network)

    def run(self):
        if self.network.install():
            self.GUI.install()

    def __del__(self):
        del self.network
        del self.GUI

if __name__ == '__main__':
    run_args = sys.argv[1:]

    port = 8000
    server_ip = 'localhost:8001'

    c = ClientApp(port, server_ip)
    c.run()

from tkinter import *


class ClientAppGUI:

    def __init__(self, network):

        def count_callback(expr):
```

```
            result_field.delete(1.0, END)
            result_field.insert(1.0, "Processing")
            resp = network.send_to_server(expr)
            if resp:
                result_field.delete(1.0, END)
                result_field.insert(1.0, resp)

        window = Tk()

        # инициализация виджетов
        expr_input = Entry(window, width=300, bd=2)
        count = Button(window, text="Посчитать", command=lambda: count_callback(expr_input.get()))
        result_label = Label(window, text="Результат выражения:")
        result_field = Text(window, width=300, bd=2)

        # инсталяция виджетов
        expr_input.pack()
        count.pack()
        result_label.pack()
        result_field.pack()

        # конфигурация окна
        window.title("Текстовый калькулятор")
        w = window.winfo_screenwidth()  # ширина экрана
        h = window.winfo_screenheight()  # высота экрана
        w = w // 2  # середина экрана
        h = h // 2
        w = w - 200  # смещение от середины
        h = h - 200
        window.geometry('400x400+{}+{}'.format(w, h))

        self.window = window

    def install(self):
        self.window.mainloop()

import socket
class ClientAppNetwork:

    def __init__(self, port, server_ip):
        addr = server_ip.split(":")
        self.server_addr = addr[0], int(addr[1])
        self.host = "localhost"
        self.port = port
        self.socket = None
        self.conn = None

    def install(self):

        try:
            self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.socket.settimeout(2)
        except socket.error:
            print("Ошибка создания сокета")
            return False

        try:
            self.conn = self.socket.connect(self.server_addr)
        except socket.error:
            print("Ошибка подключения к серверу")
            self.socket.close()
```

```
            self.socket = None
            return False
        print("Подключен")
        return True

    def send_to_server(self, data: str) -> str or None:

        if self.socket:
            try:
                self.socket.send(data.encode())
                print("Отправлено", data)
            except socket.timeout:
                return "Ошибка отправки"
            try:
                response = self.socket.recv(1024)
                print("Получено", response.decode())
                return response.decode()
            except socket.timeout:
                print("Большое время отклика")

        else:
            return None

    def __del__(self):
        self.send_to_server('finish')
        if self.conn is not None:
            self.conn.close()
        if self.socket is not None:
            self.socket.close()
            print("CLOSED")

import sys
import socketserver
from server_logic import Parser


class TextCalculatorServerHandler(socketserver.BaseRequestHandler):

    def handle(self):
        p = Parser()

        close = False
        while not close:
            data = self.request.recv(1024).decode()
            print("Получено", data)
            # прислали finish - заканчиваем цикл обработки соединения
            if "finish" in data:
                close = True
                print("Подключение завершено")
                continue
            resp = p.process(data)
            print(resp)
            try:
                self.request.sendall(resp.encode())
            except BrokenPipeError:
                continue
            except ConnectionResetError:
                continue


if __name__ == '__main__':
```

```
    args = sys.argv[1:]
    port = 8001


    addr = ("localhost", port)

    # создаем TCP сервер для получения

    with socketserver.TCPServer(addr, TextCalculatorServerHandler) as server:
        print("Сервер начал работу на {x[0]}:{x[1]}".format(x=server.server_address))
        try:
            server.serve_forever()
        except:
            print("\nСервен остановлен")
            server.shutdown()
            server.server_close()

import re


class Parser:

    def __init__(self):
        self.expr_list = []
        regexp = "[()]|[0-7]+\.[0-7]+|[0-7]+|[+]|[\-]|[*]|[/]|[\^]|sin|cos"
        self.regexp = re.compile(regexp)

    def _parse(self, data: str):
        self.expr_list = re.findall(self.regexp, data)

    def _check_parentheses(self) -> bool:
        return self.expr_list.count('(') == self.expr_list.count(')')

    def _check_illegal_characters(self, data):
        return len("".join(self.expr_list)) == len(data)


    def _calculate(self, data: str) -> str:
        operations = {
            r'sin': 'math.sin',
            r'cos': 'math.cos',
            r'\^': '**'
        }

        number = "[0-7]+\.[0-7]+|[0-7]+"
        # производим подстановку выражений
        for regexp, repl in operations.items():
            data = re.sub(regexp, repl, data)

        data = re.sub(number, self.convert_float_to_decimal, data)


        try:
            out = eval(data)
            print('Отправлено:', out)
            return self.convert_decimal_to_oct(out)

        except ZeroDivisionError as error:
            return "Деление на 0"


    def process(self, data) -> str:
```

```
    error_response = ""
    success_check = True
    self._parse(data)
    if not self._check_parentheses():
        error_response += "Непарные скобки\n"
        success_check = False
    if not self._check_illegal_characters(data):
        error_response += "Ошибка распознавания выражения\n"
        success_check = False

    if success_check:
        resp = self._calculate(data)
        return resp
    else:
        return error_response


def convert_float_to_decimal(self, number):
    number = number[0]

    number = number.split('.')
    number_integer_part = number[0]

    # Расчет целой части
    integer_results = 0
    for each_digit in enumerate(reversed(number_integer_part)):
        result = int(each_digit[1]) * (8 ** each_digit[0])
        print("{} * {}^{} = {}".format(
            each_digit[1], 8, each_digit[0], result))

        integer_results += result

    final_result = integer_results

    if len(number) > 1:
        number_float_part = number[1]

    # Расчет дробной части

        negative_length = (0 - len(number_float_part))
        number_float_part = number_float_part[::-1]


        number_float_part_reversed = []
        for number in number_float_part:
            number_float_part_reversed.append(number)

        float_results = 0
        for index in range(negative_length, 0):
            result = int(number_float_part_reversed[index]) * (8 ** index)
            print("{} * {}^{} = {}".format(
                number_float_part_reversed[index], 8, index, result))
            float_results += result

        final_result += float_results


    return str(final_result)

def convert_decimal_to_oct(self, f, n=4):
```

```
whole = int(f)
rem = (f - whole) * 8
int_ = int(rem)
rem = (rem - int_) * 8
octals = [str(abs(int_))]
count = 1

while rem and count < n:
   count += 1
   int_ = int(rem)
   rem = (rem - int_) * 8
   octals.append(str(abs(int_)))
return "{:o}.{}".format(whole, "".join(octals))
```