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))