

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

Факультет инфокоммуникационных технологий

Дисциплина:
«Web программирование»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

Выполнил:
студент группы К33392
Кононов Степан

Санкт-Петербург
2023 г.

Лабораторная №1

Задание 1

Реализовать клиентскую и серверную часть приложения. Клиент отправляет серверу сообщение «Hello, server». Сообщение должно отразиться на стороне сервера. Сервер в ответ отправляет клиенту сообщение «Hello, client». Сообщение должно отобразиться у клиента.

- Обязательно использовать библиотеку `socket`
- Реализовать с помощью протокола UDP

server.py

```
import socket

def main():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    server_address = ('localhost', 12345)

    server_socket.bind(server_address)

    print('Сервер запущен и ждет сообщений от клиента...')

    try:
        while True:
            data, client_address = server_socket.recvfrom(1024)
            message = data.decode('utf-8')
            print(f'Получено сообщение от клиента: "{message}"')

            response_message = 'Hello, client'
            server_socket.sendto(response_message.encode('utf-8'), client_address)
            print('Отправлено сообщение клиенту: "Hello, client"')
    except KeyboardInterrupt:
        print('Сервер остановлен.')

if __name__ == "__main__":
    main()
```

client.py

```
import socket

def main():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    server_address = ('localhost', 12345)

    message = 'Hello, server'

    try:

        client_socket.sendto(message.encode('utf-8'), server_address)
```

```

        print(f'Отправлено сообщение серверу: "{message}"')

        response, _ = client_socket.recvfrom(1024)
        response_message = response.decode('utf-8')
        print(f'Получен ответ от сервера: "{response_message}"')
    except Exception as e:
        print(f'Произошла ошибка: {e}')
    finally:

        client_socket.close()

if __name__ == "__main__":
    main()

```

Результат работы

```

Сервер запущен и ждет сообщений от клиента...
Получено сообщение от клиента: "Hello, server"
Отправлено сообщение клиенту: "Hello, client"

```

```

Отправлено сообщение серверу: "Hello, server"
Получен ответ от сервера: "Hello, client"

```

Задание 2

Реализовать клиентскую и серверную часть приложения. Клиент запрашивает у сервера выполнение математической операции, параметры, которые вводятся с клавиатуры. Сервер обрабатывает полученные данные и возвращает результат клиенту.

Варианты:

1. Теорема Пифагора
2. Решение квадратного уравнения.
3. Поиск площади трапеции.
4. Поиск площади параллелограмма.

Обязательно использовать библиотеку `socket`

Реализовать с помощью протокола TCP

server.py

```

import math
import socket

def solve_quadratic(a, b, c):
    discriminant = b ** 2 - 4 * a * c
    if discriminant > 0:
        x1 = (-b + math.sqrt(discriminant)) / (2 * a)
        x2 = (-b - math.sqrt(discriminant)) / (2 * a)
        return "Два корня: x1 = {}, x2 = {}".format(x1, x2)
    elif discriminant == 0:
        x = -b / (2 * a)
        return "Один корень: x = {}".format(x)
    else:
        return "Нет действительных корней"

```

```

def start_server(host, port):
    server_address = (host, port)
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(server_address)
    server_socket.listen(1)
    print(f"Сервер запущен на {host}:{port}")

    while True:
        print("Ожидание подключения клиента...")
        client_socket, client_address = server_socket.accept()
        print(f"Подключение от {client_address}")

        try:
            data = client_socket.recv(1024).decode()
            if not data:
                break

            a, b, c = map(float, data.split())
            result = solve_quadratic(a, b, c)
            client_socket.send(result.encode())
        except Exception as e:
            print(f"Произошла ошибка: {str(e)}")
        finally:
            client_socket.close()

def main():
    server_host = '127.0.0.1'
    server_port = 12345
    start_server(server_host, server_port)

if __name__ == "__main__":
    main()

```

client.py

```

import socket

def main():
    server_host = '127.0.0.1'
    server_port = 12345

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((server_host, server_port))

    try:
        a = float(input("Введите a: "))
        b = float(input("Введите b: "))
        c = float(input("Введите c: "))

        data = f"{a} {b} {c}"
        client_socket.send(data.encode())

        result = client_socket.recv(1024).decode()
        print(f"Результат: {result}")
    except Exception as e:
        print(f"Произошла ошибка: {str(e)}")
    finally:
        client_socket.close()

```

```
if __name__ == "__main__":  
    main()
```

Результат работы

```
Сервер запущен на 127.0.0.1:12345  
Ожидание подключения клиента...  
Подключение от ('127.0.0.1', 52712)  
Ожидание подключения клиента...  
Подключение от ('127.0.0.1', 52755)  
Ожидание подключения клиента...  
Подключение от ('127.0.0.1', 52762)  
Ожидание подключения клиента...
```

```
Введите a: 1  
Введите b: 8  
Введите c: 15  
Результат: Два корня: x1 = -3.0, x2 = -5.0
```

Задание 3

Реализовать серверную часть приложения. Клиент подключается к серверу. В ответ клиент получает http-сообщение, содержащее html-страницу, которую сервер подгружает из файла index.html.

server.py

```
import socket  
  
HDRS_404 = 'HTTP/1.1 404 Not Found\r\nContent-Type: text/html; charset=utf-8\r\n\r\n'  
HDRS_200 = 'HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=utf-8\r\n\r\n'  
  
SERVER_PORT = 2000  
  
def start_server():  
    print("Server is starting...")  
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
  
    try:  
        server.bind(('127.0.0.1', SERVER_PORT))  
        server.listen(1)  
        print(f'Server is listening on port {SERVER_PORT}')  
  
        while True:  
            print("Waiting for incoming connection...")  
            client_socket, address = server.accept()  
            print(f"Accepted connection from {address}")  
  
            try:  
                handle_client_request(client_socket)  
            except Exception as e:  
                print(f"Error handling client request: {e}")  
  
    except KeyboardInterrupt:  
        server.close()
```

```

        print("Server shutdown")
    except Exception as e:
        print(f"Server error: {e}")

def handle_client_request(client_socket):
    data = client_socket.recv(1024).decode('utf-8')
    content = load_page_from_get_request(data)
    client_socket.send(content)
    client_socket.shutdown(socket.SHUT_WR)
    client_socket.close()

def load_page_from_get_request(request_data):
    path = request_data.split(' ')[1]
    try:
        with open('views' + path, 'rb') as file:
            response = file.read()
            return HDRS_200.encode('utf-8') + response
    except FileNotFoundError:
        return (HDRS_404 + "Sorry...").encode('utf-8')

if __name__ == '__main__':
    start_server()

```

index.html

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Welcome</title>
  <style>
    body {
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
      font-family: "Helvetica", sans-serif;
      flex-direction: column;
      background-color: #ededed;
    }

    .container {
      display: flex;
      flex-direction: column;
      align-items: flex-start;
      padding: 20px;
    }

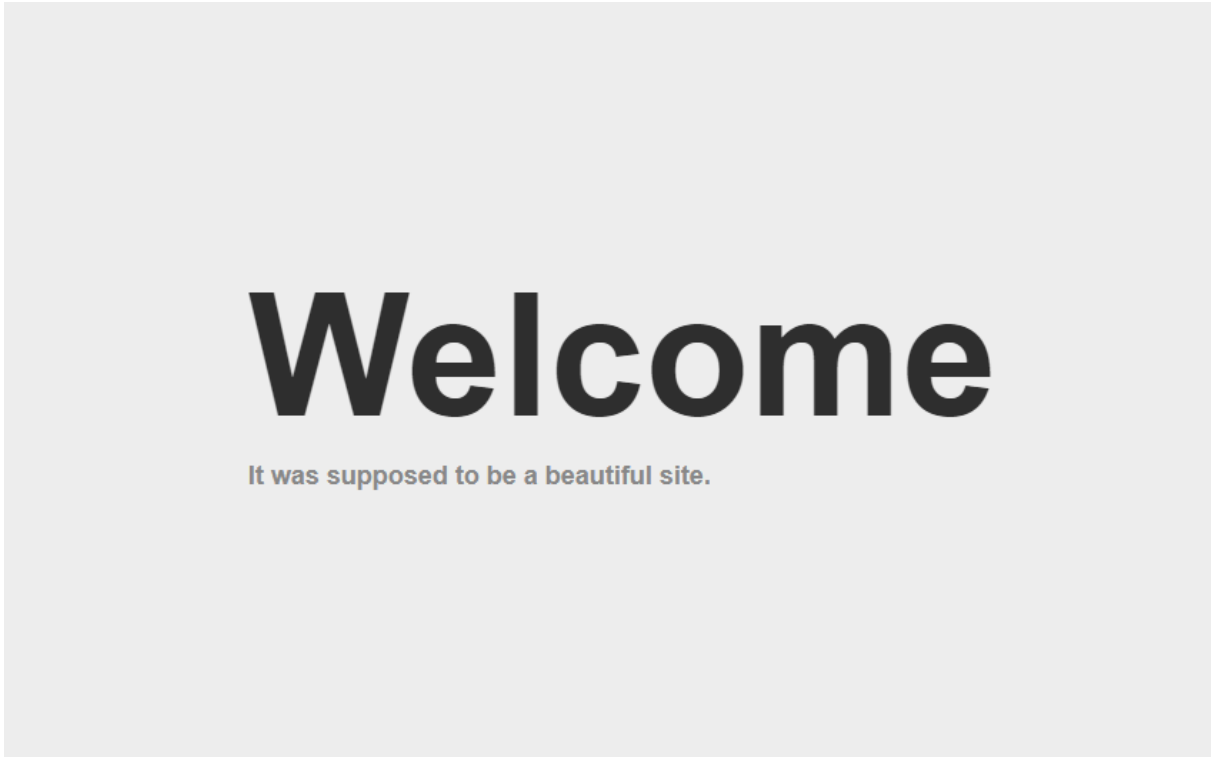
    h1 {
      font-size: 120px;
      color: #2e2e2e;
      margin: 0;
    }

    h2 {
      font-size: 18px;
      color: #888;
      margin-top: 5px;
    }
  </style>
</head>

```

```
<body>
  <div class="container">
    <h1>Welcome</h1>
    <h2>It was supposed to be a beautiful site.</h2>
  </div>
</body>
</html>
```

Результат работы



Welcome

It was supposed to be a beautiful site.

Задание 4

Реализовать двухпользовательский или многопользовательский чат. Реализация многопользовательского чата позволяет получить максимальное количество баллов.

Обязательно использовать библиотеку `threading`.

Для реализации с помощью UDP, `threading` использовать для получения сообщений у клиента.

Для применения с TCP необходимо запускать клиентские подключения и прием и отправку сообщений всем юзерам на сервере в потоках. Не забудьте сохранять юзеров, чтобы потом отправлять им сообщения.

server.py

```
import socket
import threading

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```

server_socket.bind(("127.0.0.1", 5555))
server_socket.listen(5)

clients = {}
addresses = {}

def broadcast(message, sender_name):
    for client, client_name in clients.items():
        if client_name != sender_name:
            try:
                client.send(message.encode('utf-8'))
            except:
                remove(client)

def remove(client):
    if client in clients:
        client_name = clients[client]
        del clients[client]
        del addresses[client]
        print(f"{client_name} покинул чат")
        broadcast(f"{client_name} покинул чат", None)

def handle_client(client):
    try:
        client_name = client.recv(1024).decode('utf-8')
        clients[client] = client_name
        addresses[client] = client.getpeername()
        print(f"{client_name} присоединился к чату")
        broadcast(f"{client_name} присоединился к чату", None)
    except:
        return

    while True:
        try:
            message = client.recv(1024).decode('utf-8')
            if not message:
                break
            if message.startswith("@"):
                recipient, message = message.split(" ", 1)
                recipient = recipient[1:]
                if recipient in clients.values():
                    recipient_socket = [client for client, name in clients.items() if name == recipient][0]
                    recipient_socket.send(f"{clients[client]} (личное сообщение): {message}".encode('utf-8'))
                else:
                    print(f"{clients[client]}: {message}")
                    broadcast(f"{clients[client]}: {message}", clients[client])
            except:
                remove(client)

        remove(client)

    while True:
        client_socket, client_address = server_socket.accept()
        print(f"Подключено: {client_address}")

        client_thread = threading.Thread(target=lambda: handle_client(client_socket))
        client_thread.start()

```

client.py


```

import socket
import threading

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(("127.0.0.1", 5555))

client_name = input("Введите ваше имя: ")
client_socket.send(client_name.encode('utf-8'))

def send_message():
    while True:
        message = input()
        if message.lower() == 'exit':
            client_socket.close()
            print("Выход из чата")
            break
        client_socket.send(message.encode('utf-8'))

def receive_message():
    while True:
        try:
            message = client_socket.recv(1024).decode('utf-8')
            print(message)
        except:
            client_socket.close()
            print("Сервер отключился")
            break

send_thread = threading.Thread(target=send_message)
receive_thread = threading.Thread(target=receive_message)

send_thread.start()
receive_thread.start()

```

Результат работы

- client 1

```

Введите ваше имя: Степан
Степан присоединился к чату
Соня присоединился к чату
Артур 42 присоединился к чату
Форд присоединился к чату
Артур 42: А скоро ли конец света?
Форд: Примерно через 20 минут
@Соня Как жизнь?
Всем пока
Соня: Пока

```

- client 2

```
Введите ваше имя: Соня
Соня присоединился к чату
Артур 42 присоединился к чату
Форд присоединился к чату
Артур 42: А скоро ли конец света?
Форд: Примерно через 20 минут
Степан (личное сообщение): Как жизнь?
Степан: Всем пока
Пока
```

- client 3

```
Введите ваше имя: Артур 42
Артур 42 присоединился к чату
Форд присоединился к чату
А скоро ли конец света?
Форд: Примерно через 20 минут
Степан: Всем пока
Соня: Пока
```

- client 4

```
Введите ваше имя: Форд
Форд присоединился к чату
Артур 42: А скоро ли конец света?
Примерно через 20 минут
Степан: Всем пока
Соня: Пока
```

- server

```
Подключено: ('127.0.0.1', 53169)
Степан присоединился к чату
Подключено: ('127.0.0.1', 53170)
Соня присоединился к чату
Подключено: ('127.0.0.1', 53171)
Артур 42 присоединился к чату
Подключено: ('127.0.0.1', 53178)
Форд присоединился к чату
Артур 42: А скоро ли конец света?
Форд: Примерно через 20 минут
Степан: Всем пока
Соня: Пока
```

Задание 5

Необходимо написать простой web-сервер для обработки GET и POST http запросов средствами Python и библиотеки socket.

Задание: сделать сервер, который может:

- Принять и записать информацию о дисциплине и оценке по дисциплине.
- Отдать информацию обо всех оценках по дисциплине в виде html-страницы.

server.py

```
import socket
from urllib.parse import parse_qs

class MyHTTPServer:
    def __init__(self, host, port, name):
        self.host = host
        self.port = port
        self.name = name
        self.data = {}

    def serve_forever(self):
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_socket.bind((self.host, self.port))
        server_socket.listen(1)
        print(f"Server is listening on {self.host}:{self.port}")

        try:
            while True:
                client_socket, client_address = server_socket.accept()
                self.serve_client(client_socket)
        except KeyboardInterrupt:
            server_socket.close()
            print("Server shutdown")

    def serve_client(self, client_socket):
        request_data = client_socket.recv(1024).decode('utf-8')
        method, url, _ = self.parse_request(request_data)
        body = request_data.split('\r\n')[-1]

        if method == 'GET':
            if url == '/grades':
                response = self.get_grades()
            else:
                response = "HTTP/1.1 404 Not Found\n\nPage not found"
        elif method == 'POST':
            if url == '/add_grade':
                response = self.add_grade(body)
            else:
                response = "HTTP/1.1 404 Not Found\n\nPage not found"
        else:
            response = "HTTP/1.1 405 Method Not Allowed\n\nMethod not allowed"

        client_socket.sendall(response.encode('utf-8'))
        client_socket.close()

    def parse_request(self, request_data):
        lines = request_data.split('\n')
        method, url, _ = lines[0].split()
        return method, url, _

    def handle_request(self, method, url, headers):
        if method == 'GET':
            if url == '/grades':
                return self.get_grades()
            else:
                return "HTTP/1.1 404 Not Found\n\nPage not found"
        elif method == 'POST':
            if url == '/add_grade':
                return self.add_grade(headers)
```

```

        else:
            return "HTTP/1.1 404 Not Found\n\nPage not found"
    else:
        return "HTTP/1.1 405 Method Not Allowed\n\nMethod not allowed"

def add_grade(self, body):
    discipline, grade = self.parse_body_add_grade(body)

    if discipline and grade:
        self.data[discipline] = grade
        return "HTTP/1.1 200 OK\n\nGrade added successfully"
    else:
        return "HTTP/1.1 400 Bad Request\n\nInvalid request parameters"

def parse_body_add_grade(self, body):
    parsed_body = parse_qs(body)

    discipline = parsed_body.get('Discipline', [None])[0]
    grade = parsed_body.get('Grade', [None])[0]

    return discipline, grade

def get_grades(self):
    response = "HTTP/1.1 200 OK\nContent-Type: text/html\n\n"
    response += "<html><body><h1>Grades</h1>"
    for discipline, grade in self.data.items():
        response += f"<p>{discipline}: {grade}</p>"
    response += "</body></html>"
    return response

if __name__ == '__main__':
    host = "localhost"
    port = 8080
    name = "MyHTTPServer"
    serv = MyHTTPServer(host, port, name)
    try:
        serv.serve_forever()
    except KeyboardInterrupt:
        pass

```

Результат работы

The screenshot shows a web browser interface for a POST request to `http://localhost:8080/add_grade`. The 'Body' tab is selected, showing form data with two fields: 'Discipline' (Biology) and 'Grade' (90).

Key	Value
<input checked="" type="checkbox"/> Discipline	Biology
<input checked="" type="checkbox"/> Grade	90
Key	Value

HTTP

http://localhost:8080/grades

GET

⌵

http://localhost:8080/grades

Params

Authorization

Headers (7)

Body

Pre-request Script

Query Params

	Key
	Key

Body

Cookies

Headers (1)

Test Results

Pretty

Raw

Preview

Visualize

Grades

Biology: 90

Math: 100