

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Web-программирование

Отчет

ЛАБОРАТОРНАЯ РАБОТА №1

Работа с сокетами

Выполнила: Пронина

Александра

Группа K33392

Проверил:

Говоров А. И.

Санкт-Петербург

2023 г.

Цель: овладеть практическими навыками и умениями реализации web-серверов и использования сокетов.

Оборудование: компьютерный класс.

Программное обеспечение: Python 2.7–3.6, библиотеки Python: sys, socket.

Задание 1:

Здесь код представляет собой простой пример клиент-серверного взаимодействия с использованием протокола UDP.

Укажем пункты-пояснения того, что происходит на стороне сервера:

1. Импортируется модуль `socket`.
2. Определяется функция `main()`, которая будет выполнять основную логику сервера.
3. Задается IP-адрес и порт сервера.
4. Создается сокет сервера с помощью функции `socket.socket()`, указывая параметры `AF_INET` (IPv4) и `SOCK_DGRAM` (UDP).
5. Связывается сокет сервера с заданным IP-адресом и портом с помощью метода `bind()`.
6. Выводится сообщение о том, что сервер слушает на заданном IP-адресе и порту.
7. В бесконечном цикле сервер принимает сообщение от клиента с помощью метода `recvfrom()`, указывая максимальный размер сообщения в байтах.
8. Выводится сообщение от клиента, преобразованное из байтовой строки в строку с помощью метода `decode()`.
9. Создается ответное сообщение для клиента.
10. Ответное сообщение отправляется клиенту с помощью метода `sendto()`, преобразованное в байтовую строку с помощью метода `encode()`.

Укажем пункты-пояснения того, что происходит на стороне клиента:

1. Импортируется модуль `socket`.

2. Определяется функция `main()`, которая будет выполнять основную логику клиента.
3. Задается IP-адрес и порт сервера.
4. Задается сообщение, которое будет отправлено серверу.
5. Создается сокет клиента с помощью функции `socket.socket()`, указывая параметры `AF_INET` (IPv4) и `SOCK_DGRAM` (UDP).
6. Отправляется сообщение серверу с помощью метода `sendto()`, указывая сообщение в виде байтовой строки и адрес сервера.
7. Получается ответное сообщение от сервера с помощью метода `recvfrom()`, указывая максимальный размер сообщения в байтах.
8. Выводится ответное сообщение от сервера, преобразованное из байтовой строки в строку с помощью метода `decode()`.
9. Закрывается сокет клиента. После определения функции `main()` на стороне клиента, проверяется, что код выполняется как самостоятельный скрипт, а не импортирован как модуль, и вызывается функция `main()`.

```
import socket

def main():
    server_ip = "127.0.0.1"
    server_port = 1245
    message = "Hello, server"

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    client_socket.sendto(message.encode(), (server_ip, server_port))

    response, _ = client_socket.recvfrom(1024)
    print("Response from server:", response.decode())

    client_socket.close()

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

```
import socket

def main():
    server_ip = "127.0.0.1"
    server_port = 12345

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.bind((server_ip, server_port))

    print("Server is listening on", server_ip, "port", server_port)
```

```

while True:
    message, client_address = server_socket.recvfrom(1024)
    print("Message from client:", message.decode())

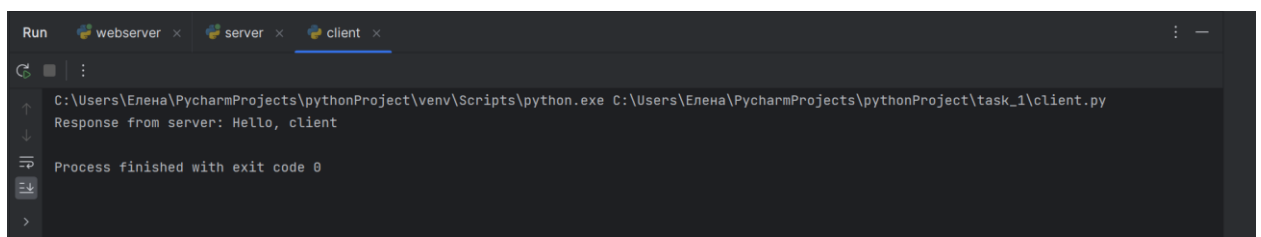
    response = "Hello, client"
    server_socket.sendto(response.encode(), client_address)

if __name__ == "__main__":
    main()

```

Проверяем работу:

Для этого сначала запустим серверную часть, а затем клиентскую:



И клиент получает такое сообщение от сервера.

Задание 2:

На стороне сервера:

1. Импортируется модуль `socket`.
2. Импортируется модуль `math` для выполнения математических операций.
3. Определяется функция `process_request()`, которая будет выполнять основную логику сервера. В этой функции происходит обработка различных операций (a, b, c, d) с заданными параметрами.
4. Создается сокет сервера с помощью функции `socket.socket()`, указывая параметры `AF_INET` (IPv4) и `SOCK_STREAM` (TCP).
5. Задается IP-адрес и порт сервера.
6. Связывается сокет сервера с заданным IP-адресом и портом с помощью метода `bind()`.
7. Сокет сервера начинает слушать входящие подключения с помощью метода `listen()`.
8. В бесконечном цикле сервер ожидает подключения клиента с помощью метода `accept()`. Когда клиент подключается, создается новый сокет клиента и адрес клиента.
9. Получается запрос от клиента с помощью метода `recv()`, указывая максимальный размер сообщения в байтах, и декодируется из байтовой строки в строку.
10. Запрос разбивается на операцию и параметры с помощью метода `split()`.

11. Вызывается функция `process_request()` для обработки запроса и получения результата.
12. Результат преобразуется в строку и отправляется клиенту с помощью метода `send()`, преобразовав его в байтовую строку.
13. Сокет клиента закрывается.

На стороне клиента:

1. Импортируется модуль `socket`.
2. Определяется функция `send_request()`, которая будет выполнять основную логику клиента. В этой функции происходит отправка запроса на сервер и получение ответа.
3. Создается сокет клиента с помощью функции `socket.socket()`, указывая параметры `AF_INET` (IPv4) и `SOCK_STREAM` (TCP).
4. Задается IP-адрес и порт сервера.
5. Клиенту предлагается выбрать код операции и параметры, которые будут отправлены на сервер.
6. Устанавливается соединение с сервером с помощью метода `connect()`, указывая адрес сервера.
7. Запрос формируется в виде строки, кодируется в байтовую строку и отправляется серверу с помощью метода `send()`.
8. Получается ответ от сервера с помощью метода `recv()`, указывая максимальный размер сообщения в байтах, и декодируется из байтовой строки в строку.
9. Сокет клиента закрывается. После определения функций на стороне клиента, проверяется, что код выполняется как самостоятельный скрипт, а не импортирован как модуль, и вызывается функция `main()`.

```
import socket

def send_request(operate, params):

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    server_address = ('localhost', 12)
    client_socket.connect(server_address)

    request = operate + ' ' + ' '.join(params)
    client_socket.send(request.encode())

    response = client_socket.recv(1024).decode()

    client_socket.close()

    return response

operation = input("Введите код операции (a, b, c или d): ")
parameters = input("Введите параметры через пробел: ").split()
```

```
result = send_request(operation, parameters)

print("Результат:", result)
```

```
import socket
import math

def process_request(code_operation, params):
    if code_operation == 'a':
        a = float(params[0])
        b = float(params[1])
        c = math.sqrt(a ** 2 + b ** 2)
        return str(c)
    elif code_operation == 'b':
        a = float(params[0])
        b = float(params[1])
        c = float(params[2])
        d = b ** 2 - 4 * a * c
        if d > 0:
            x1 = (-b + math.sqrt(d)) / (2 * a)
            x2 = (-b - math.sqrt(d)) / (2 * a)
            return "x1 = " + str(x1) + ", x2 = " + str(x2)
        elif d == 0:
            x = -b / (2 * a)
            return "x = " + str(x)
        else:
            return "Уравнение не имеет корней, принадлежащих области действительных чисел"
    elif code_operation == 'c':
        a = float(params[0])
        b = float(params[1])
        h = float(params[2])
        s = (a + b) * h / 2
        return str(s)
    elif code_operation == 'd':
        a = float(params[0])
        h = float(params[1])
        s = a * h
        return str(s)
    else:
        return "Несуществующая операция"

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

server_address = ('localhost', 12)
server_socket.bind(server_address)

server_socket.listen(1)

while True:
    print("Ожидание подключения клиента...")
    client_socket, client_address = server_socket.accept()

    print("Подключение от", client_address)

    request = client_socket.recv(1024).decode()
    operation, *parameters = request.split()

    result = process_request(operation, parameters)
```

```

client_socket.send(result.encode())

client_socket.close()

```

Project: pythonProject

- task_1
 - client.py
 - server.py
- task_2 (selected)
 - client.py
 - server.py
- task_3
 - client.py
 - index.html
 - server.py
- task_4
- task_5
- webserver.py

Run: server (1) × client (1) ×

```

C:\Users\Елена\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\Елена\PycharmProjects\pythonProject\task_2\client.py
Введите код операции (a, b, c или d): a
Введите параметры через пробел: 3 4
Результат: 5.0
Process finished with exit code 0

```

pythonProject > task_2 > client.py

Project: pythonProject

- task_1
 - client.py
 - server.py
- task_2 (selected)
 - client.py
 - server.py
- task_3
 - client.py
 - index.html
 - server.py
- task_4
- task_5

Run: server (1) × client (1) ×

```

C:\Users\Елена\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\Елена\PycharmProjects\pythonProject\task_2\client.py
Введите код операции (a, b, c или d): c
Введите параметры через пробел: 1 2 3
Результат: 4.5
Process finished with exit code 0

```

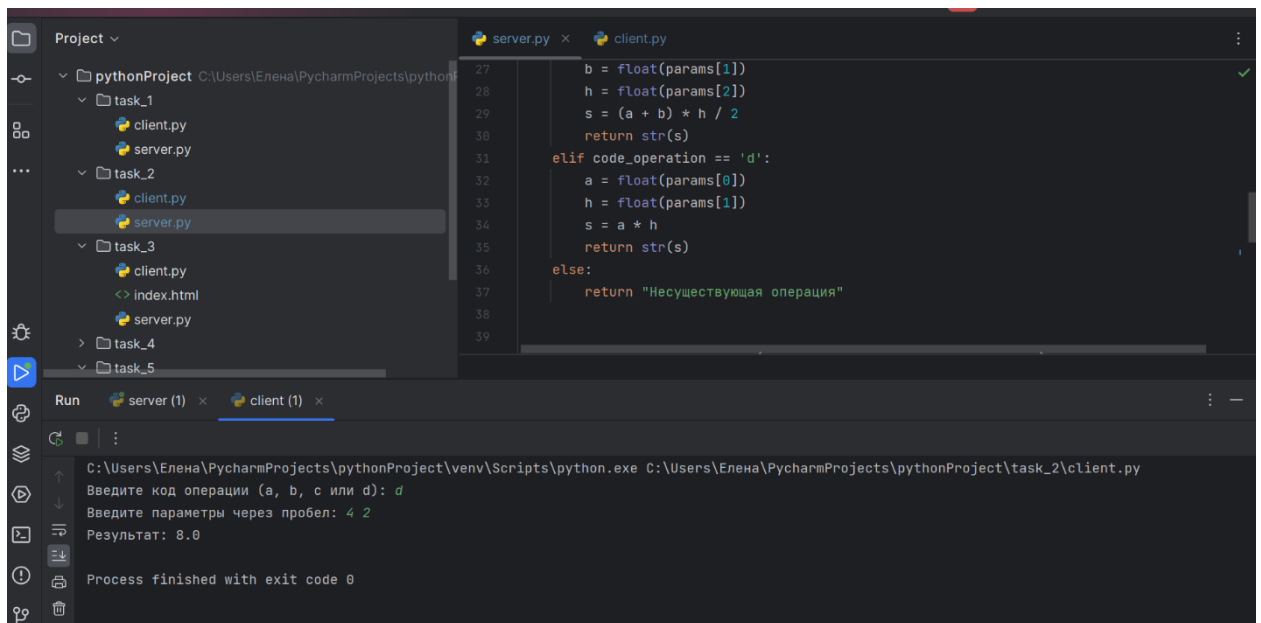
pythonProject > task_2 > client.py

Run: server (1) × client (1) ×

```

C:\Users\Елена\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\Елена\PycharmProjects\pythonProject\task_2\client.py
Введите код операции (a, b, c или d): b
Введите параметры через пробел: 1 2 1
Результат: x = -1.0
Process finished with exit code 0

```



Задание 3:

Код представляет собой пример простого веб-сервера.

На стороне сервера: Импортируется модуль `socket`. Затем определяется функция `process_request()`, которая будет выполнять основную логику сервера. В данном случае функция открывает файл `"index.html"` и читает его содержимое. Затем формируется HTTP-ответ с кодом 200 OK и типом контента `text/html`, добавляется содержимое файла и возвращается ответ. После создается сокет сервера с помощью функции `socket.socket()`, указывая параметры `AF_INET` (IPv4) и `SOCK_STREAM` (TCP).

Задается IP-адрес и порт сервера. Сокет сервера связывается с заданным IP-адресом и портом с помощью метода `bind()` и начинает слушать входящие подключения с помощью метода `listen()`. В бесконечном цикле сервер ожидает подключения клиента с помощью метода `accept()`. Когда клиент подключается, создается новый сокет клиента и адрес клиента. Принимается запрос от клиента с помощью метода `recv()`, указывая максимальный размер сообщения в байтах, и декодируется из байтовой строки в строку. Вызывается функция `process_request()` для обработки запроса и получения результата. Результат преобразуется в байтовую строку и отправляется клиенту с помощью метода `send()`. Сокет клиента закрывается.

На стороне клиента: Сперва импортируется модуль `socket`. После создается сокет клиента с помощью функции `socket.socket()`, указывая параметры `AF_INET` (IPv4) и `SOCK_STREAM` (TCP).

Задается IP-адрес и порт сервера и формируется запрос в виде строки, содержащей HTTP-запрос на получение главной страницы сайта "localhost". Устанавливается соединение с сервером с помощью метода `connect()`, указывая адрес сервера. Запрос кодируется в байтовую строку и отправляется серверу с помощью метода `send()`. Получается ответ от сервера с помощью метода `recv()`, указывая максимальный размер сообщения в байтах, и декодируется из байтовой строки в строку. Ответ выводится на экран. Сокет клиента закрывается.

В данном коде сервер просто возвращает содержимое файла "index.html" при получении любого запроса от клиента. Клиент отправляет HTTP-запрос на получение главной страницы сайта и выводит ответ на экран.

```
import socket

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_address = ('localhost', 1234)
client_socket.connect(server_address)

request = "GET / HTTP/1.1\r\nHost: localhost\r\n\r\n"
client_socket.send(request.encode())

response = client_socket.recv(1024).decode()

print(response)

client_socket.close()
```

```
<!DOCTYPE html>
<html lang="en">
<head><title> Page</title></head>
<body>
<h1>Welcome!</h1>
<p>Test page</p>
</body>
</html>
```

```
import socket

def process_request():
    with open('index.html', 'r') as file:
        html = file.read()

    response = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n" + html

    return response

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

server_address = ('localhost', 1234)
```

```

server_socket.bind(server_address)

server_socket.listen(1)

while True:
    print("Ожидание подключения клиента...")
    client_socket, client_address = server_socket.accept()

    print("Подключение от", client_address)

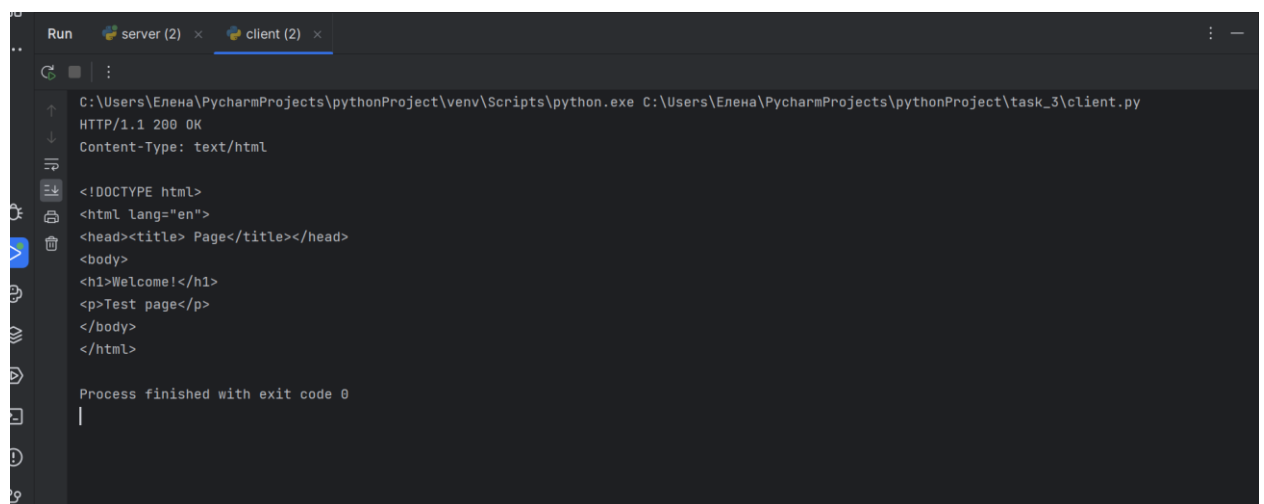
    request = client_socket.recv(1024).decode()

    response = process_request()

    client_socket.send(response.encode())

    client_socket.close()

```



```

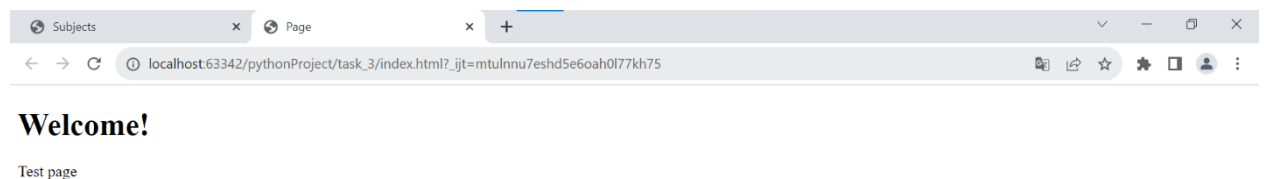
Run
server (2) x client (2) x
C:\Users\Елена\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\Елена\PycharmProjects\pythonProject\task_3\client.py
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html>
<html lang="en">
<head><title> Page</title></head>
<body>
<h1>Welcome!</h1>
<p>Test page</p>
</body>
</html>

Process finished with exit code 0

```

Клиентская часть написана в доказательство, что клиент увидит сайт.



Задание 4:

Здесь задача была создать много пользовательский чат. Для этого определяем модуль сервера и нескольких клиентов.

```

import socket
import threading

from task_4.server import ChatServer

class ChatClient:
    def __init__(self):
        self.host = 'localhost'
        self.port = 456
        self.client_socket = None

    def start(self):
        nickname = input("Enter your nickname: ")

        self.client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.client_socket.connect((self.host, self.port))
        self.client_socket.sendall(nickname.encode())

        receive_thread = threading.Thread(target=self.receive_messages)
        receive_thread.start()

        while True:
            message = input()
            self.client_socket.sendall(message.encode())

    def receive_messages(self):
        while True:
            try:
                message = self.client_socket.recv(1024).decode()
                if message:
                    print(message)
                else:
                    break
            except:
                break

if __name__ == "__main__":
    mode = input("Choose mode (server/client): ")

    if mode == "server":
        server = ChatServer()
        server.start()
    elif mode == "client":
        client = ChatClient()
        client.start()
    else:
        print("Invalid mode")

```

```

import socket
import threading

class ChatServer:
    def __init__(self):
        self.host = 'localhost'
        self.port = 456
        self.server_socket = None
        self.clients = []

    def start(self):

```

```

        self.server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.server_socket.bind((self.host, self.port))
        self.server_socket.listen(5)

        print("Chat server started on {}:{}".format(self.host, self.port))

        while True:
            client_socket, address = self.server_socket.accept()
            client_thread = threading.Thread(target=self.handle_client,
args=(client_socket,))
            client_thread.start()

        def handle_client(self, client_socket):
            nickname = client_socket.recv(1024).decode()
            print("{} connected".format(nickname))

            self.broadcast("{} joined the chat".format(nickname))

            self.clients.append((nickname, client_socket))

            while True:
                try:
                    message = client_socket.recv(1024).decode()
                    if message:
                        self.broadcast("{}: {}".format(nickname, message))
                    else:
                        self.remove_client(nickname, client_socket)
                        break
                except:
                    self.remove_client(nickname, client_socket)
                    break

            def remove_client(self, nickname, client_socket):
                client_socket.close()
                self.clients.remove((nickname, client_socket))
                self.broadcast("{} left the chat".format(nickname))

            def broadcast(self, message):
                for client in self.clients:
                    client[1].sendall(message.encode())

```

Задание 5:

Данный код представляет собой простой HTTP-сервер, который может принимать POST-запросы и GET-запросы.

1. Импортируется класс `BaseHTTPRequestHandler` и класс `HTTPServer` из модуля `http.server`.
2. Создается класс `MyServer`, который наследуется от `BaseHTTPRequestHandler`. В этом классе определены два метода: `do_POST` и `do_GET`.
3. Метод `do_POST` используется для обработки POST-запросов. Сначала получается длина тела запроса из заголовка `Content-Length`. Затем данные запроса считываются из потока

чтения `rfile` и декодируются из байтовой строки в строку с помощью метода `decode('utf-8')`.

4. Полученные данные разделяются на две части: дисциплину и оценку. Для этого используется метод `split('&')`, который разделяет строку по символу `'&'`. Затем каждая часть разделяется по символу `'='` с помощью метода `split('=')[1]`.

5. Данные записываются в словарь `self.data`. Если дисциплина уже существует в словаре, то оценка добавляется в список оценок для этой дисциплины. Если дисциплины нет в словаре, то создается новая запись с ключом дисциплины и значением - список оценок.

6. После записи данных в словарь, сервер отправляет успешный ответ с кодом 200 и заголовком `'Content-type', 'text/html'`. Затем в поток записи `wfile` отправляется строка `'Данные записаны'`.

7. Метод `do_GET` используется для обработки GET-запросов. Сначала создается HTML-страница, которая содержит информацию об оценках по каждой дисциплине. Для этого происходит итерация по словарю `self.data`. Для каждой дисциплины создается заголовок `h3`, затем создается маркированный список `ul`, в котором перечисляются оценки для данной дисциплины.

8. После создания HTML-страницы, сервер отправляет успешный ответ с кодом 200 и заголовком `'Content-type', 'text/html'`. Затем в поток записи `wfile` отправляется сгенерированная HTML-страница, закодированная в байтовую строку с помощью метода `encode('utf-8')`.

9. В конце определен метод `run`, который запускает HTTP-сервер на указанном порту (по умолчанию 6332). Создается экземпляр класса `HTTPServer` с указанными адресом сервера и классом обработчика запросов. Затем вызывается метод `serve_forever()` для бесконечного обслуживания запросов.

10. В конце кода вызывается функция `run()` для запуска сервера.

```
from http.server import BaseHTTPRequestHandler, HTTPServer

class MyServer(BaseHTTPRequestHandler):
    # Словарь для хранения информации о дисциплине и оценках
    data = {}

    def do_POST(self):
        # Получение данных из POST-запроса
        content_length = int(self.headers['Content-Length'])
        post_data = self.rfile.read(content_length).decode('utf-8')

        # Разделение данных на дисциплину и оценку
        discipline, grade = post_data.split('&')
        discipline = discipline.split('=')[1]
```

```

        grade = grade.split('=')[1]

        # Запись данных в словарь
        if discipline not in self.data:
            self.data[discipline] = []
        self.data[discipline].append(grade)

        # Отправка успешного ответа
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()
        self.wfile.write('Данные записаны')

    def do_GET(self):
        # Создание HTML-страницы с информацией об оценках по дисциплине

        html = '<html><body>'
        for discipline, grades in self.data.items():
            html += '<h3>' + discipline + '</h3>'
            html += '<ul>'
            for grade in grades:
                html += '<li>' + grade + '</li>'
            html += '</ul>'
        html += '</body></html>'

        # Отправка HTML-страницы
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()
        self.wfile.write(html.encode('utf-8'))

def run(server_class=HTTPServer, handler_class=MyServer, port=6332):
    server_address = ('', port)
    httpd = server_class(server_address, handler_class)
    print('Сервер работает на порте', port)
    httpd.serve_forever()

run()

```

```

C:\Users\Елена\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\Елена\PycharmProjects\pythonProject\task_4\client.py
Choose mode (server/client): server
Chat server started on localhost:456
Elena connected
Sasha connected

```

