Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

по лабораторной работе № 1
«Работа с сокетами»
по дисциплине «**Web-программирование**»

Выполнила:

Микулина А.Р.

К33421

Проверил:

Говоров А.И.

Санкт-Петербург, 2023

Цель работы:

Овладеть практическими навыками и умениями реализации web-серверов и использования сокетов.

Практическое задание 1:

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

Обязательно использовать библиотеку socket

Реализовать с помощью протокола UDP

client.py:

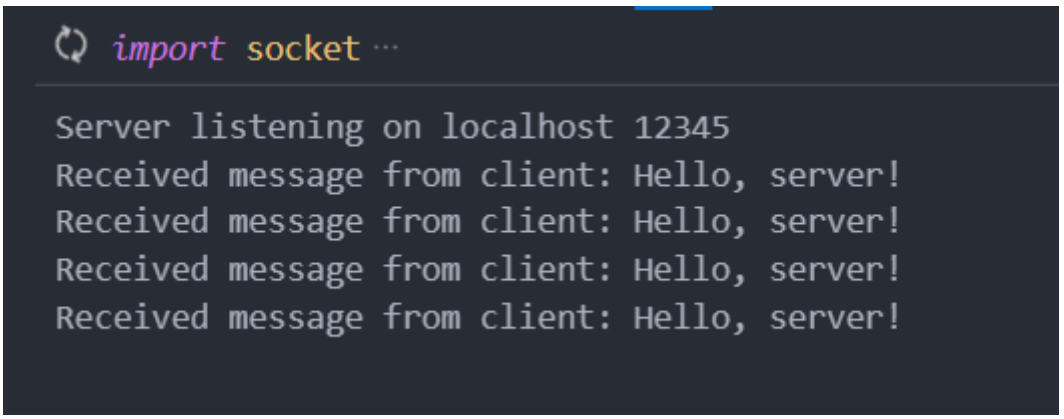
```
import socket

server_ip = 'localhost'

server_port = 12345

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

message = 'Hello, server!'

sock.sendto(message.encode(), (server_ip, server_port))

data, server_address = sock.recvfrom(4096)

response = data.decode()

print('Server response:', response)

sock.close()
```

server.py:

```
import socket

server_ip = 'localhost'

server_port = 12345
```

```python
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

sock.bind((server_ip, server_port))

print('Server listening on', server_ip, server_port)

while True:

    data, client_address = sock.recvfrom(4096)

    message = data.decode()

    print('Received message from client:', message)

    response = 'Hello, client!'

    sock.sendto(response.encode(), client_address)
```

результат:



Практическое 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. Мой вариант: Теорема Пифагора

Обязательно использовать библиотеку socket Реализовать с помощью протокола TCP

client.py:

```
import socket

def send_request(host, port, a, b, c):

    try:

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

        client_socket.connect((host, port))

        request = f"{a},{b},{c}"

        client_socket.sendall(request.encode())

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

        print(f"The result is: {result}")

        client_socket.close()

    except ConnectionRefusedError:

        print("Can't connect to the server")

def start_client():

    host = 'localhost'

    port = 12345

    a = input('''Enter the size of the sides.

                You must leave one of the values blank!\nEnter cathetus A:
''')

    a = float(a) if a else None

    b = input("Enter cathetus B: ")
```

```
    b = float(b) if b else None
    c = input("Enter hypotenuse C: ")
    c = float(c) if c else None
    send_request(host, port, a, b, c)
start_client()
```

server.py:

```
import socket
def calculate_side(a, b, c):
    if a is None:
        return (c ** 2 - b ** 2) ** 0.5
    elif b is None:
        return (c ** 2 - a ** 2) ** 0.5
    elif c is None:
        return (a ** 2 + b ** 2) ** 0.5

def handle_client(connection):
    data = connection.recv(1024).decode()
    params = data.split(',')
    if len(params) != 3:
        connection.send(b"Invalid parameters")
    else:
        try:
            a = float(params[0]) if params[0] != "None" else None
            b = float(params[1]) if params[1] != "None" else None
            c = float(params[2]) if params[2] != "None" else None
```

```
            result = calculate_side(a, b, c)
            connection.send(str(result).encode())
        except ValueError:
            connection.send(b"Invalid parameters")
    connection.close()


def start_server():
    host = 'localhost'
    port = 12345
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)
    print("Server started. Waiting for connections...")
    while True:
        client_socket, address = server_socket.accept()
        print(f"Connection established from {address[0]}:{address[1]}")
        handle_client(client_socket)
start_server()
```

результат:

```
Server started. Waiting for connections...
Connection established from 127.0.0.1:65082
```

```
Enter the size of the sides.
            You must leave one of the values blank!
Enter cathetus A: 3
Enter cathetus B: 4
Enter hypotenuse C:
The result is: 5.0
PS C:\Users\Alice\Desktop\shara\WebProgramming\Lab1>
```

Практическое 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.

Обязательно использовать библиотеку socket

client.py:

```
import socket

def get_html():

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

    client_socket.connect(('localhost', 8080))

    request = 'GET / HTTP/1.0\r\n\r\n'

    client_socket.sendall(request.encode())

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

    print(response)

    client_socket.close()

get_html()
```

server.py:

```
import socket

def send_html(client_socket):

    with open('tsk3_index.html', 'r') as file:

        response = 'HTTP/1.0 200 OK\n\n' + file.read()

        client_socket.sendall(response.encode())

def handle_client(client_socket):

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

```
    if request.startswith('GET / HTTP'):
        send_html(client_socket)
    client_socket.close()
def run_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', 8080))
    server_socket.listen(1)
    print('Server listening on port 8080...')
    while True:
        client_socket, address = server_socket.accept()
        print('Connection from', address)
        handle_client(client_socket)
run_server()
```

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Simple HTTP Server</title>
</head>
<body>
    <h1>I love myself!</h1>
</body>
</html>
```

результат:

```
import socket ···

Server listening on port 8080...
Connection from ('127.0.0.1', 50306)
```

```
HTTP/1.0 200 OK

<!DOCTYPE html>
<html>
<head>
    <title>Simple HTTP Server</title>
</head>
<body>
    <h1>I love myself!</h1>
</body>
</html>
```

Практическое задание 4:

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

Обязательно использовать библиотеку socket

Реализовано с помощью протокола TCP.

Обязательно использовать библиотеку threading.

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.

client.py:

```
import socket

import threading

server_address = input("Enter server address: ")

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

client_socket.connect((server_address, 12345))

client_name = input("Enter your name: ")

client_socket.send(client_name.encode())

def receive():

    while True:

        try:

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

            print(message)

        except:

            break

def send():

    while True:
```

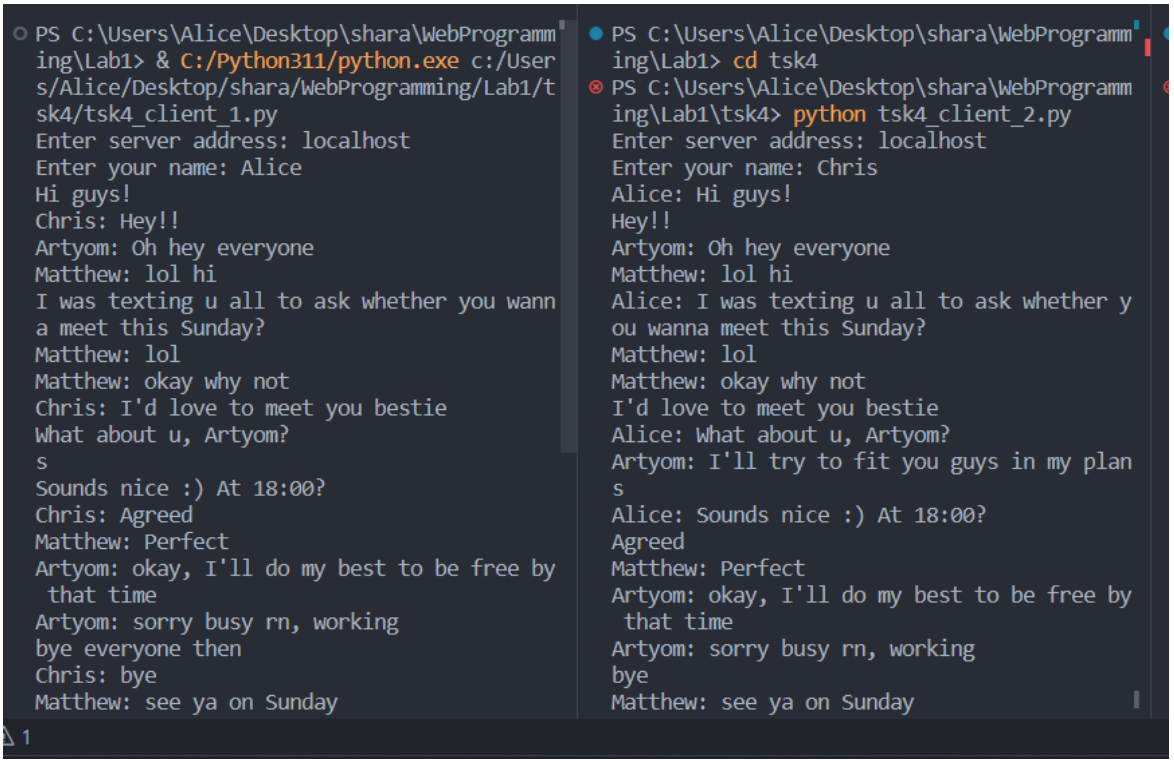
```
        message = input()

        client_socket.send(message.encode())

receive_thread = threading.Thread(target=receive)

send_thread = threading.Thread(target=send)

receive_thread.start()

send_thread.start()
```

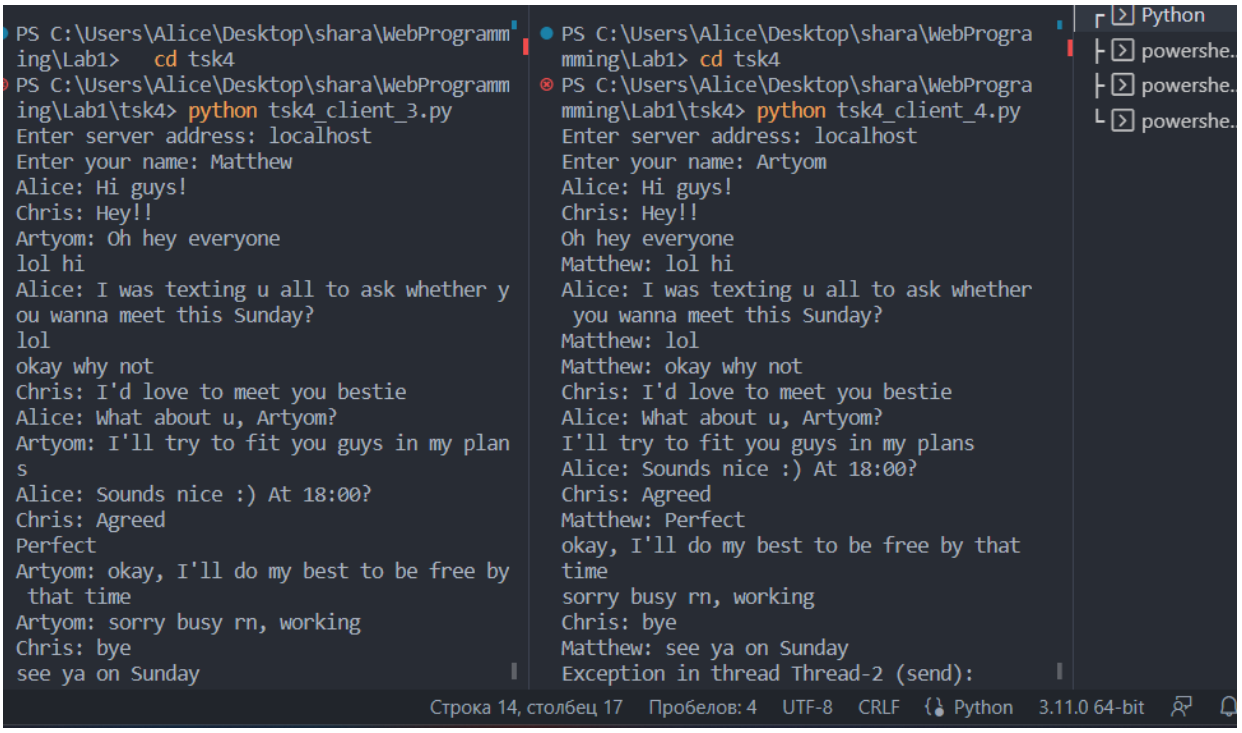
server.py:

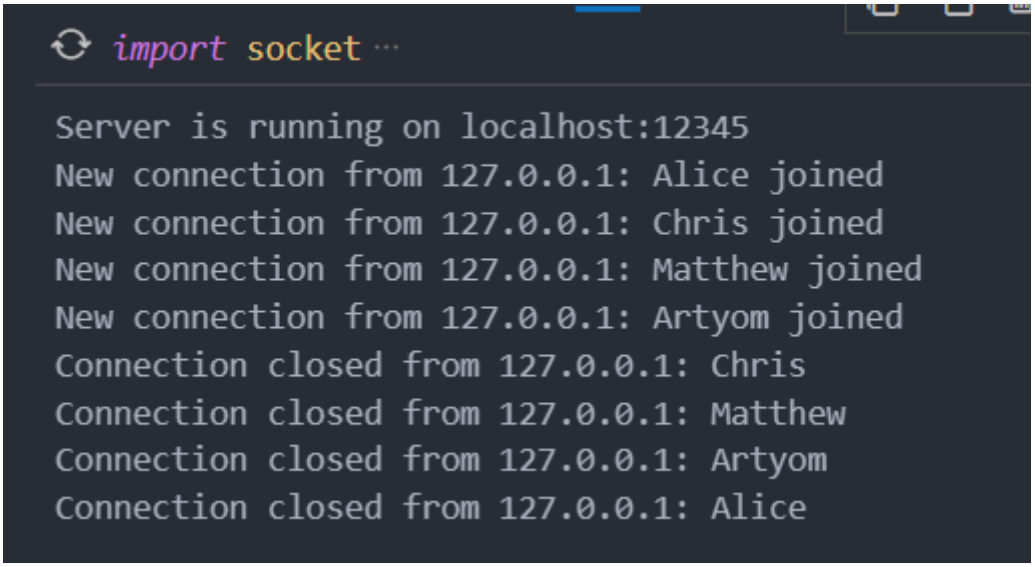
```
import socket

import threading

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

server_address = ('localhost', 12345)

server_socket.bind(server_address)

server_socket.listen(5)

print("Server is running on {}:{}".format(*server_address))

clients = []

def handle_client(client_socket, client_address):

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

     print("New connection from {}: {} joined".format(client_address[0],
client_name))

    clients.append((client_socket, client_name))

    while True:

        try:

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

            if not message:

                break
```

```
            for client in clients:
                if client[0] != client_socket:
                    client[0].send("{}: {}".format(client_name,
message).encode())
        except ConnectionResetError:
            break
    clients.remove((client_socket, client_name))
        print("Connection closed from {}: {}".format(client_address[0],
client_name))
while True:
    client_socket, client_address = server_socket.accept()
            client_thread = threading.Thread(target=handle_client,
args=(client_socket, client_address))
    client_thread.start()
```

результат:




PS C:\Users\Alice\Desktop\shara\WebProgramming\Lab1> cd tsk4
PS C:\Users\Alice\Desktop\shara\WebProgramming\Lab1\tsk4> python tsk4_client_3.py
Enter server address: localhost
Enter your name: Matthew
Alice: Hi guys!
Chris: Hey!!
Artyom: Oh hey everyone
lol hi
Alice: I was texting u all to ask whether you wanna meet this Sunday?
lol
okay why not
Chris: I'd love to meet you bestie
Alice: What about u, Artyom?
Artyom: I'll try to fit you guys in my plans
Alice: Sounds nice :) At 18:00?
Chris: Agreed
Perfect
Artyom: okay, I'll do my best to be free by that time
Artyom: sorry busy rn, working
Chris: bye
see ya on Sunday
PS C:\Users\Alice\Desktop\shara\WebProgramming\Lab1> cd tsk4
PS C:\Users\Alice\Desktop\shara\WebProgramming\Lab1\tsk4> python tsk4_client_4.py
Enter server address: localhost
Enter your name: Artyom
Alice: Hi guys!
Chris: Hey!!
Oh hey everyone
Matthew: lol hi
Alice: I was texting u all to ask whether you wanna meet this Sunday?
Matthew: lol
Matthew: okay why not
Chris: I'd love to meet you bestie
Alice: What about u, Artyom?
I'll try to fit you guys in my plans
Alice: Sounds nice :) At 18:00?
Chris: Agreed
Matthew: Perfect
okay, I'll do my best to be free by that time
sorry busy rn, working
Chris: bye
Matthew: see ya on Sunday
Exception in thread Thread-2 (send):
Python
powershe..
powershe..
powershe..
Строка 14, столбец 17
Пробелов: 4
UTF-8
CRLF
Python
3.11.0 64-bit


import socket ···
Server is running on localhost:12345
New connection from 127.0.0.1: Alice joined
New connection from 127.0.0.1: Chris joined
New connection from 127.0.0.1: Matthew joined
New connection from 127.0.0.1: Artyom joined
Connection closed from 127.0.0.1: Chris
Connection closed from 127.0.0.1: Matthew
Connection closed from 127.0.0.1: Artyom
Connection closed from 127.0.0.1: Alice

Практическое 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 оценах по дсициплине в виде html-страницы.

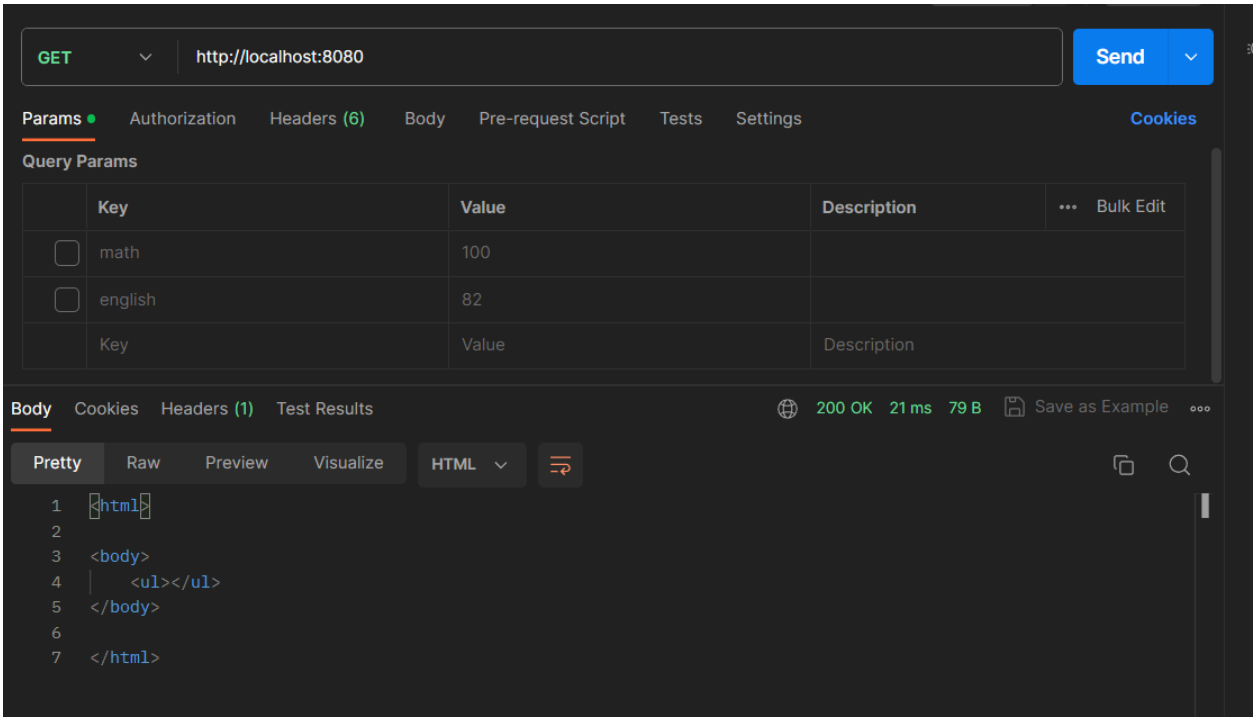
server.py:

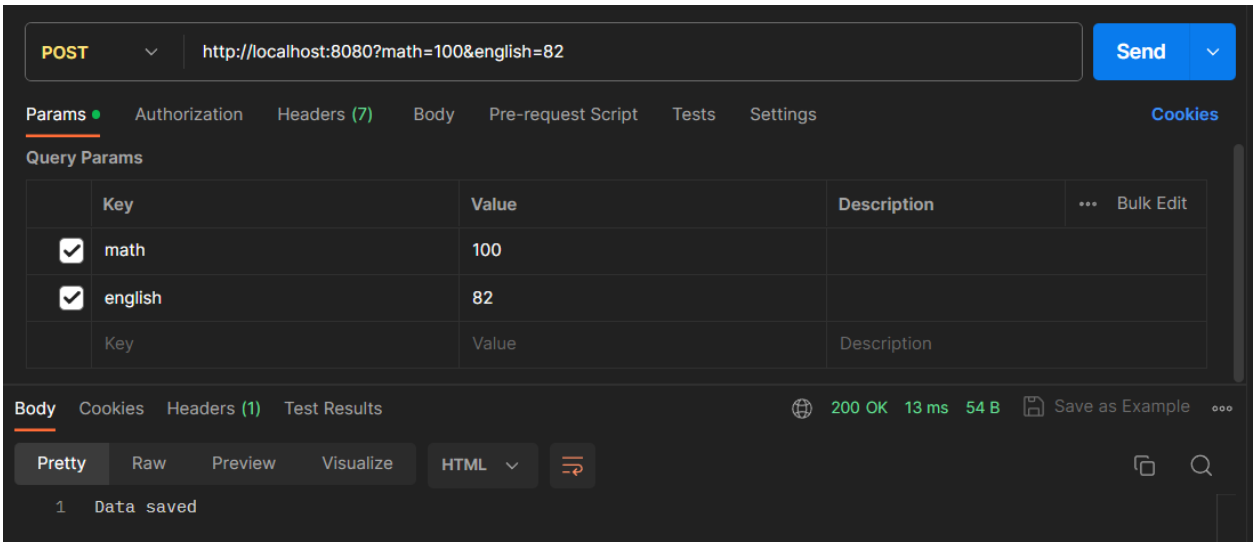
```
import socket

import sys

class MyHTTPServer:

    def __init__(self, host: str, port: int):

        self.host = host

        self.port = port

        self.connect = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        self.connect.bind((host, port))

        self.connect.listen(1)

        self.grades = {}

    def serve_forever(self):

         while True:

            client, address = self.connect.accept()

            self.serve_client(client)

    def serve_client(self, client):

        data = client.recv(16384).decode()

        self.parse_request(client, data)

    def parse_request(self, client, data):
```

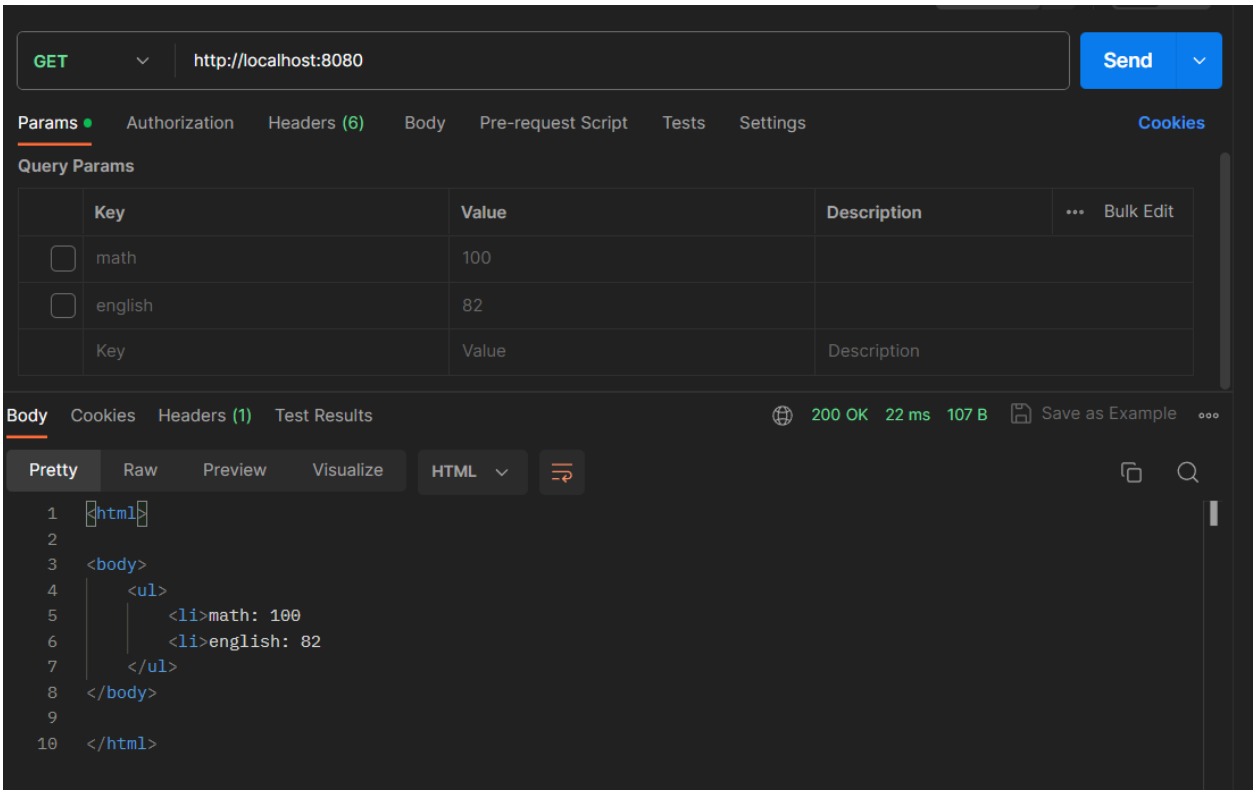
```
        lines = data.split("\n")
        method, url, version = lines[0].split()
        params = None
        if "?" in url:
            param_list = url.split("?")[1].split("&")
                    params = {p.split("=")[0]: p.split("=")[1] for p in
param_list}
        self.handle_request(client, method, params)
    def handle_request(self, client, method, params):
        if method == 'POST':
            disciplines = params.keys()
            for discipline in disciplines:
                grade = params[discipline]
                self.grades[discipline] = grade
            self.send_response(client, 200, "OK", "Data saved")
        elif method == 'GET':
                                    self.send_response(client, 200, "OK",
self.grades_to_html_page())
        else:
            self.send_response(client, 404, "Not Found", "Wrong method.")
    def send_response(self, client, code, reason, body):
                    response = f"HTTP/1.1 {code} {reason}\nContent-Type:
text/html\n\n{body}"
        client.send(response.encode())
        client.close()
    def grades_to_html_page(self):
        page = "<html><body><ul>"
```

```
        for discipline, grade in self.grades.items():
            page += f"<li>{discipline}: {grade}"
        page += "</ul></body></html>"
        return page
if __name__ == '__main__':
    host = "localhost"
    port = 8080
    serv = MyHTTPServer(host, port)
    try:
        serv.serve_forever()
    except KeyboardInterrupt:
        serv.connect.close()
```

результат:




POST
http://localhost:8080?math=100&english=82
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
math
100
english
82
Key
Value
Description
Body
Cookies
Headers (1)
Test Results
200 OK
13 ms
54 B
Save as Example
Pretty
Raw
Preview
Visualize
HTML
1 Data saved


GET
http://localhost:8080
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
math
100
english
82
Key
Value
Description
Body
Cookies
Headers (1)
Test Results
200 OK
22 ms
107 B
Save as Example
Pretty
Raw
Preview
Visualize
HTML
1 <html>
2
3 <body>
4
5 math: 100
6 english: 82
7
8 </body>
9
10 </html>