Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное образовательное учреждение высшего

образования

«Национальный исследовательский университет ИТМО»

Факультет инфокоммуникационных технологий

Лабораторная работа №1

«Работа с сокетами»

по дисциплине

«Web-программирование»

Выполнил:
Студент 3 курса ФИКТ
группы К33402
Ф.И.О. Иконенко Данил Алексеевич
Проверил:
Говоров А. И.

Санкт-Петербург

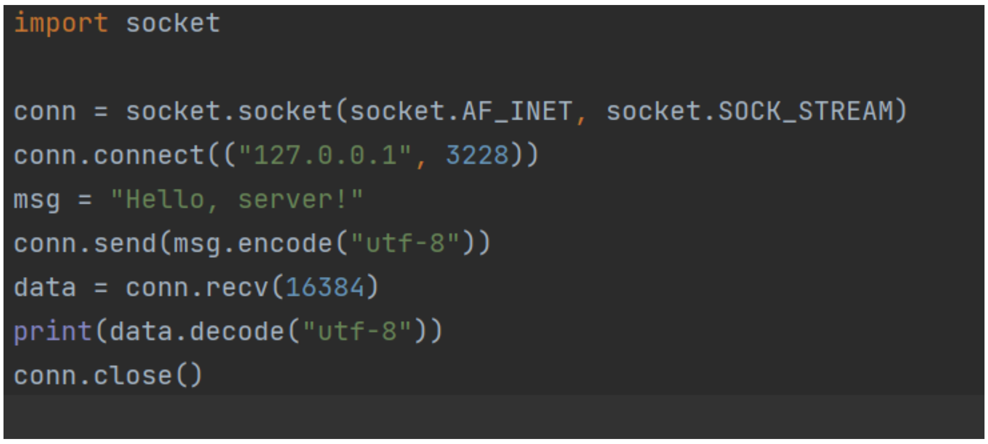
2021

Цель работы:

Реализовать клиентскую и серверную часть четырёх программ на Python, использующих сокеты.

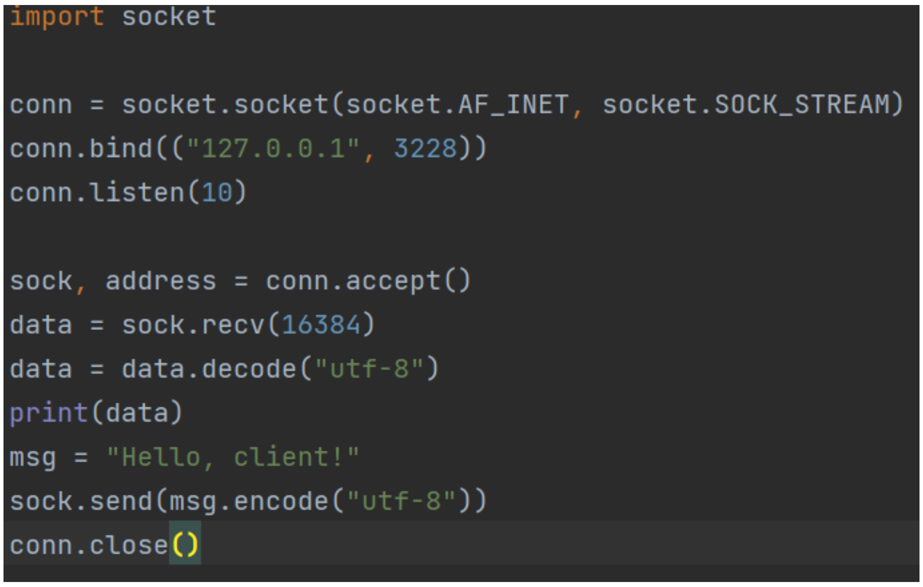
Задание 1

Клиент:



```python
import socket

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.connect(("127.0.0.1", 3228))
msg = "Hello, server!"
conn.send(msg.encode("utf-8"))
data = conn.recv(16384)
print(data.decode("utf-8"))
conn.close()
```

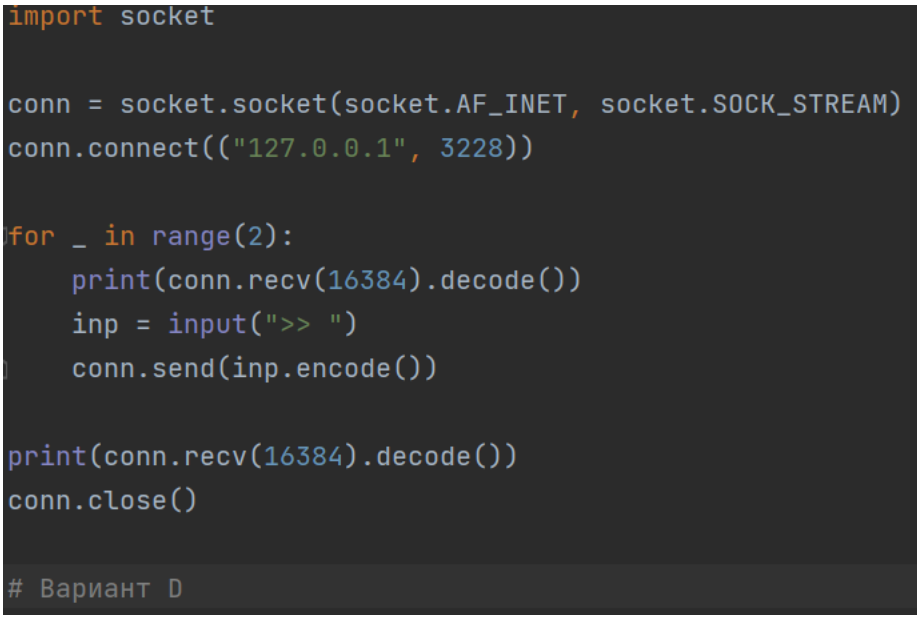
Сервер:



```python
import socket

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.bind(("127.0.0.1", 3228))
conn.listen(10)

sock, address = conn.accept()
data = sock.recv(16384)
data = data.decode("utf-8")
print(data)
msg = "Hello, client!"
sock.send(msg.encode("utf-8"))
conn.close()
```

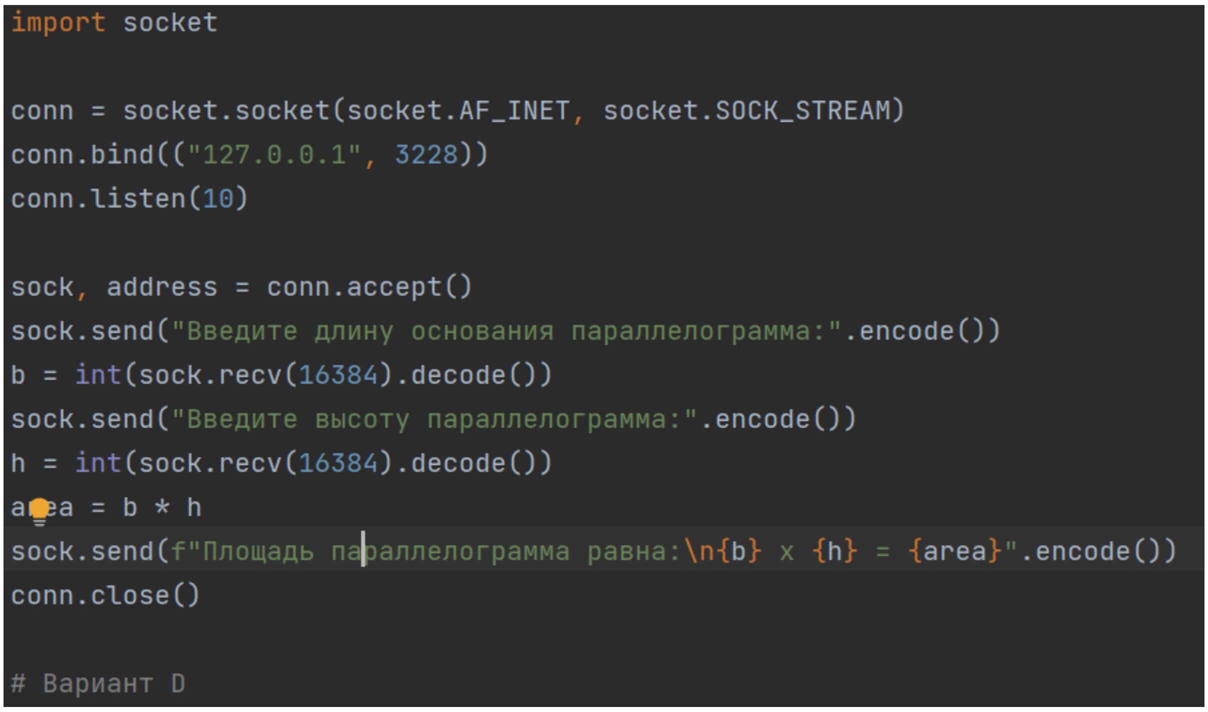
Задание 2

Клиент:



```
import socket

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.connect(("127.0.0.1", 3228))

for _ in range(2):
    print(conn.recv(16384).decode())
    inp = input(">> ")
    conn.send(inp.encode())

print(conn.recv(16384).decode())
conn.close()

# Вариант D
```

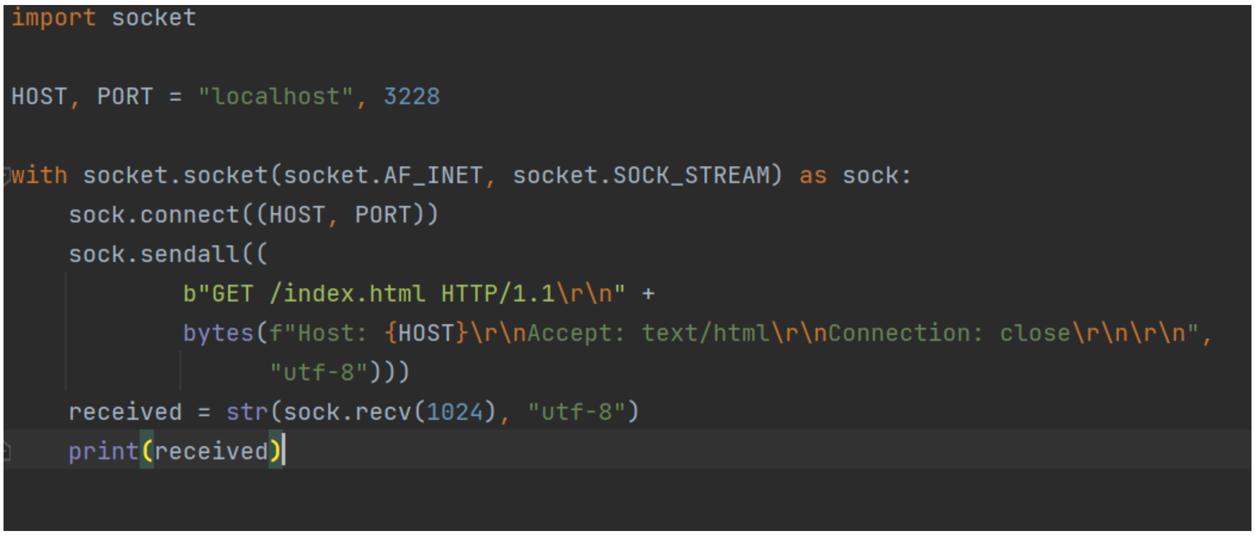
Сервер:



```
import socket

conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.bind(("127.0.0.1", 3228))
conn.listen(10)

sock, address = conn.accept()
sock.send("Введите длину основания параллелограмма:".encode())
b = int(sock.recv(16384).decode())
sock.send("Введите высоту параллелограмма:".encode())
h = int(sock.recv(16384).decode())
area = b * h
sock.send(f"Площадь параллелограмма равна:\n{b} x {h} = {area}".encode())
conn.close()

# Вариант D
```

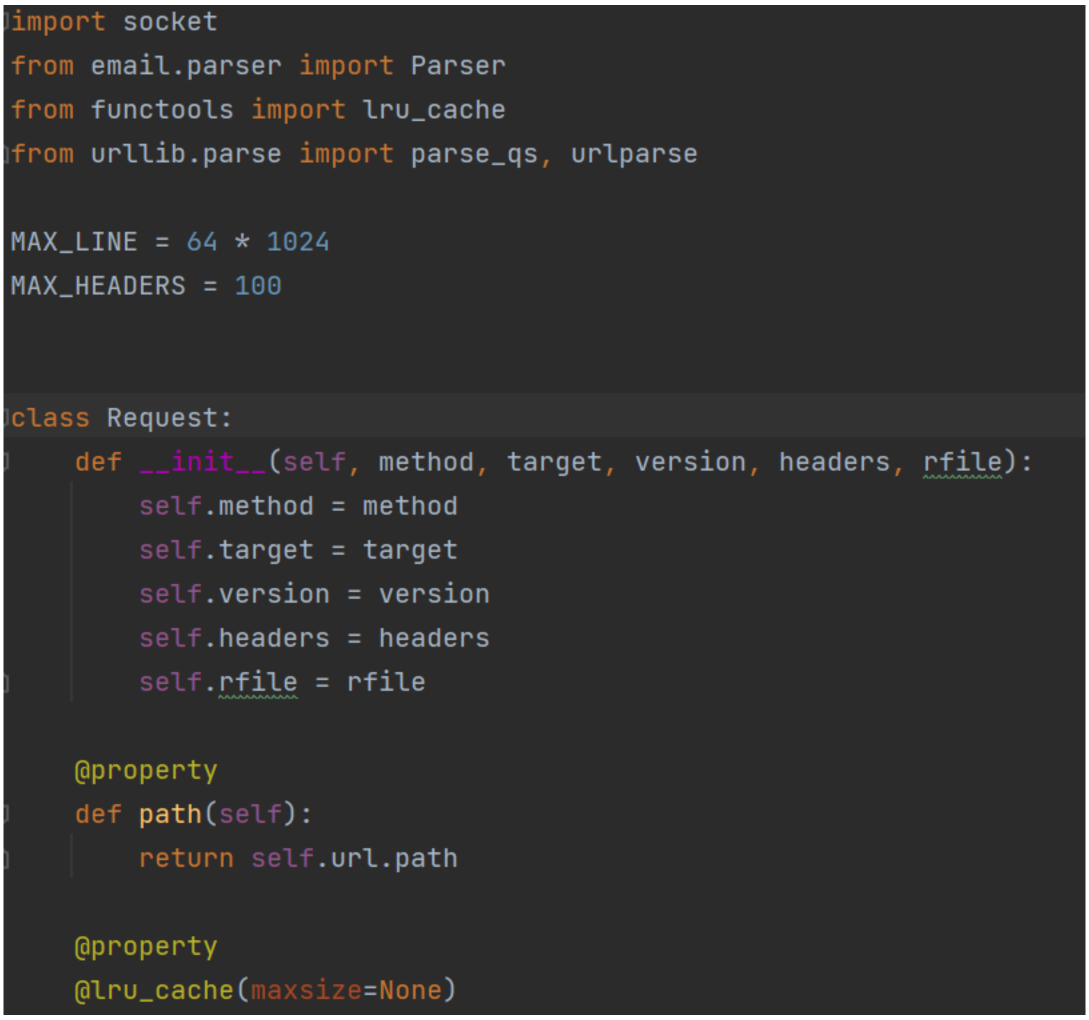
Задание 3

Клиент:

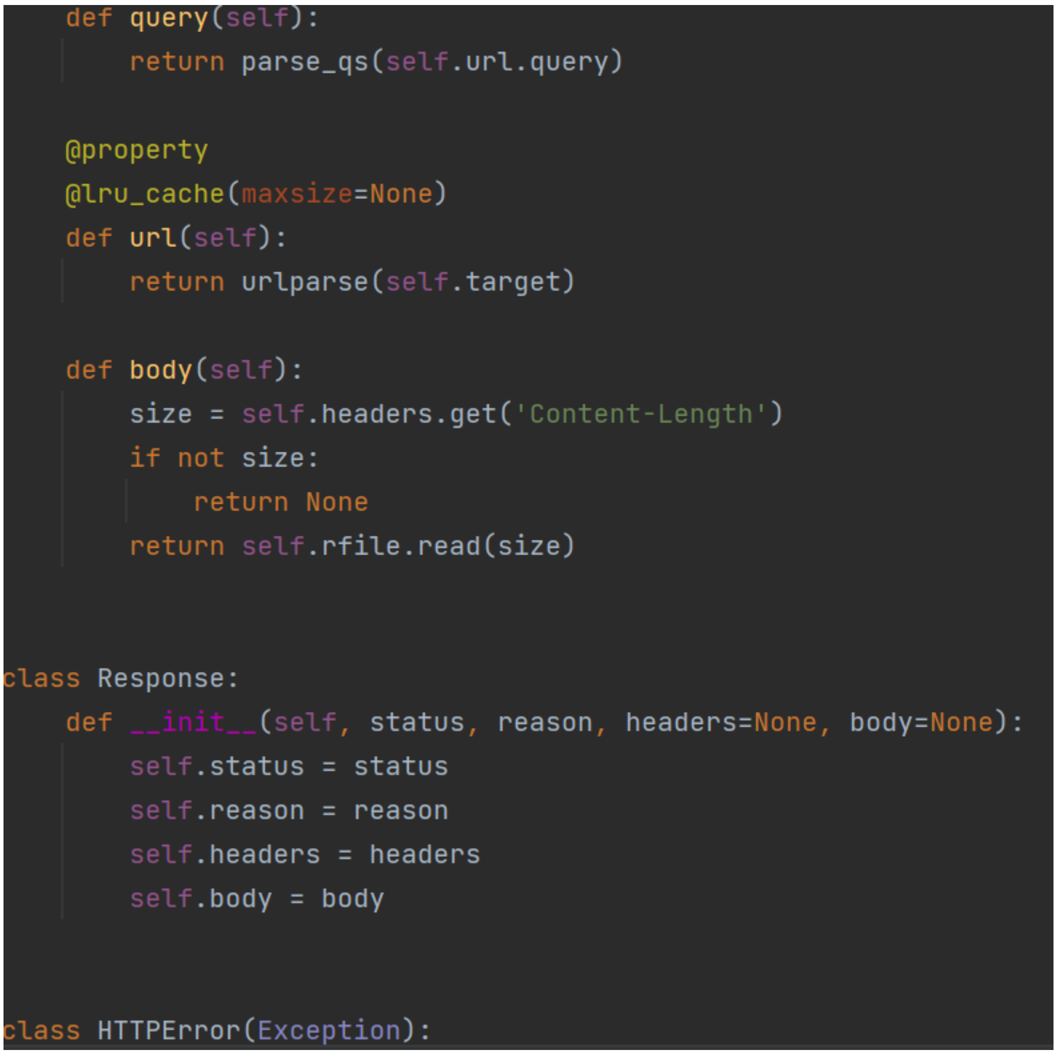


```
import socket

HOST, PORT = "localhost", 3228

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((HOST, PORT))
    sock.sendall((
            b"GET /index.html HTTP/1.1\r\n" +
            bytes(f"Host: {HOST}\r\nAccept: text/html\r\nConnection: close\r\n\r\n",
                  "utf-8")))
    received = str(sock.recv(1024), "utf-8")
    print(received)
```

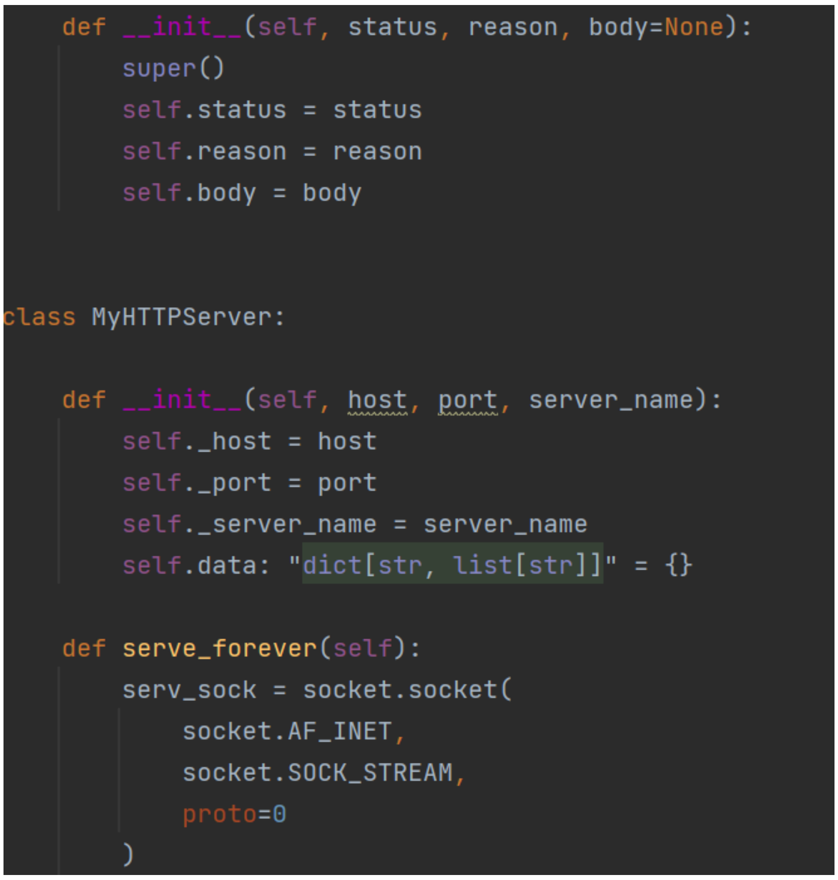
Сервер:



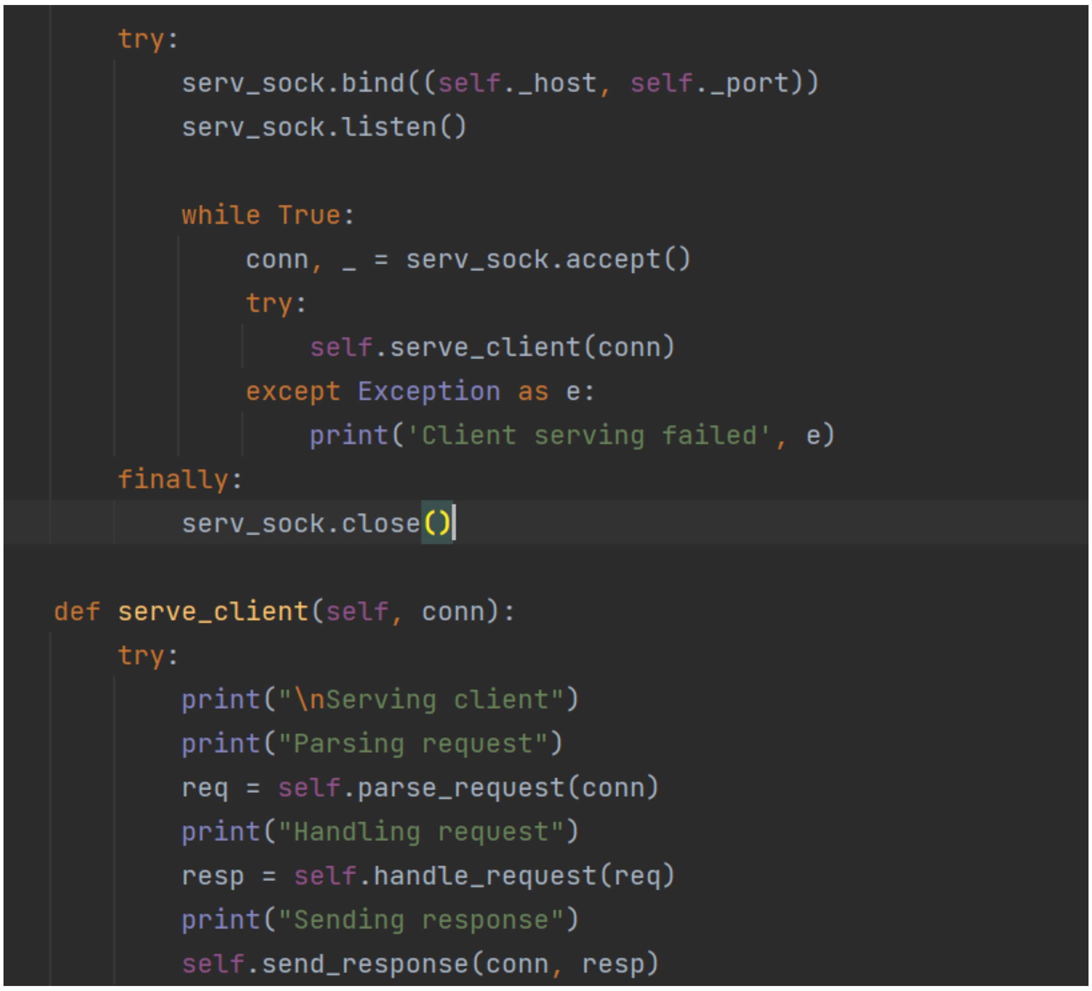
```
import socket
from email.parser import Parser
from functools import lru_cache
from urllib.parse import parse_qs, urlparse

MAX_LINE = 64 * 1024
MAX_HEADERS = 100


class Request:
    def __init__(self, method, target, version, headers, rfile):
        self.method = method
        self.target = target
        self.version = version
        self.headers = headers
        self.rfile = rfile

    @property
    def path(self):
        return self.url.path

    @property
    @lru_cache(maxsize=None)
```



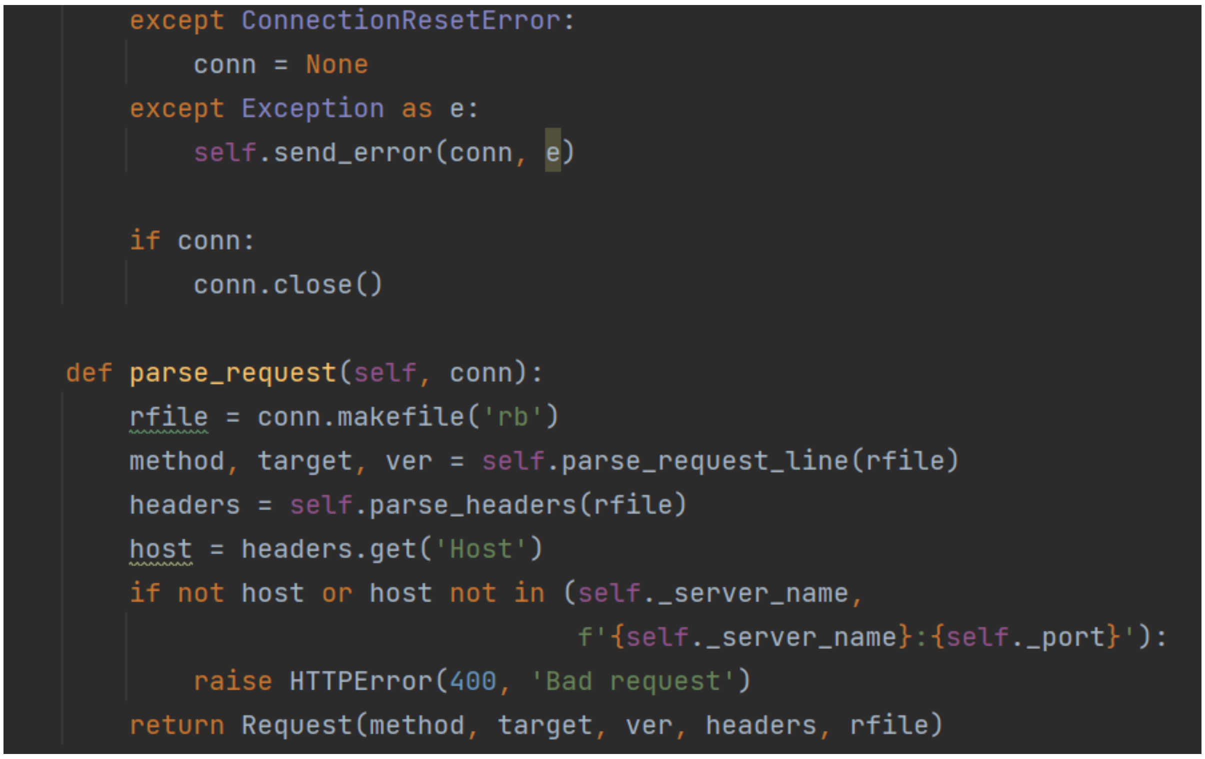
```
    def query(self):
        return parse_qs(self.url.query)

    @property
    @lru_cache(maxsize=None)
    def url(self):
        return urlparse(self.target)

    def body(self):
        size = self.headers.get('Content-Length')
        if not size:
            return None
        return self.rfile.read(size)


class Response:
    def __init__(self, status, reason, headers=None, body=None):
        self.status = status
        self.reason = reason
        self.headers = headers
        self.body = body


class HTTPError(Exception):
```



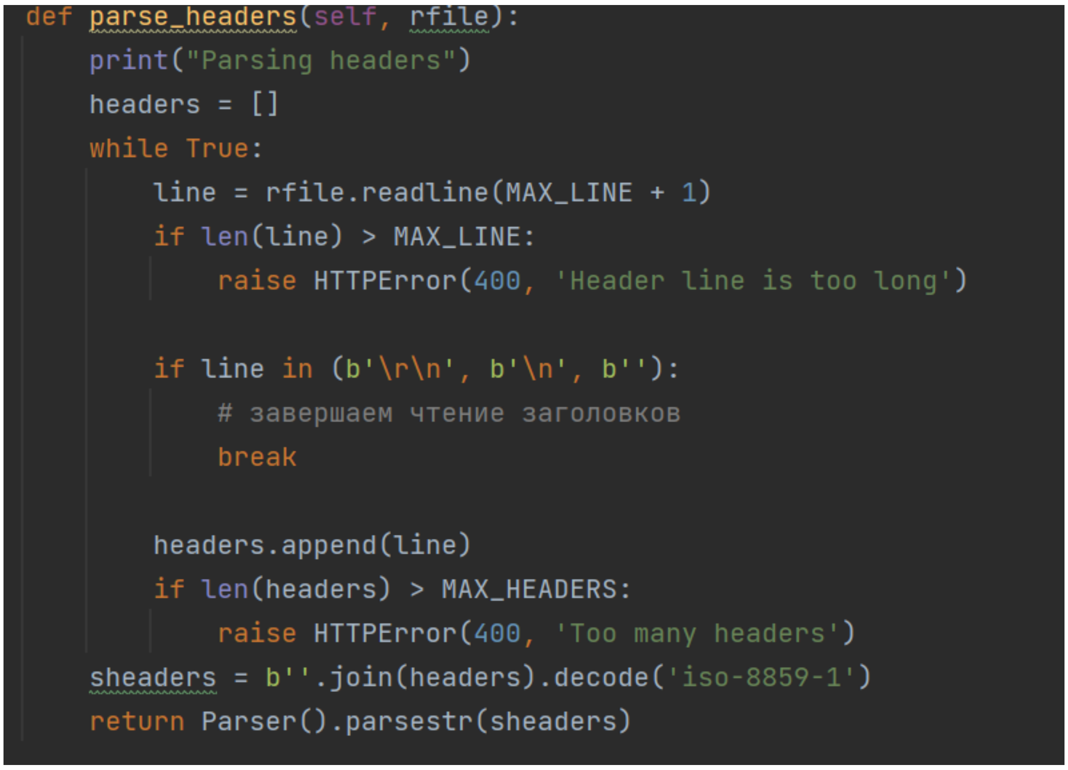
```
    def __init__(self, status, reason, body=None):
        super()
        self.status = status
        self.reason = reason
        self.body = body


class MyHTTPServer:

    def __init__(self, host, port, server_name):
        self._host = host
        self._port = port
        self._server_name = server_name
        self.data: "dict[str, list[str]]" = {}

    def serve_forever(self):
        serv_sock = socket.socket(
            socket.AF_INET,
            socket.SOCK_STREAM,
            proto=0
        )
```



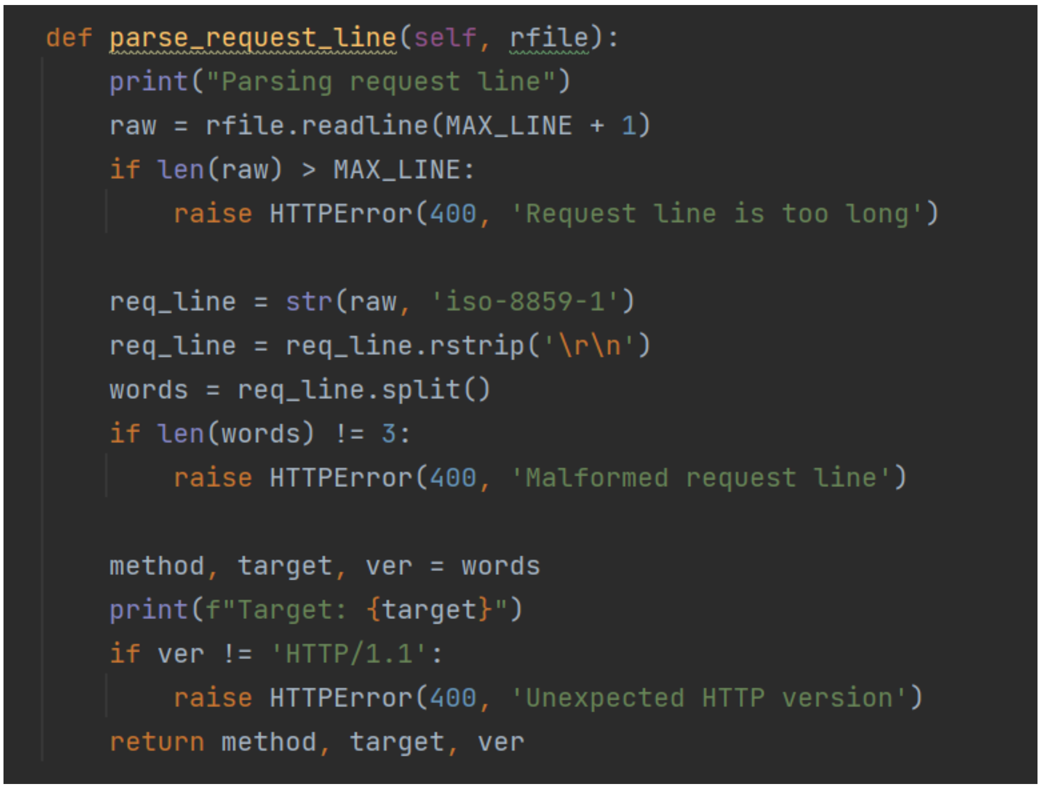
```python
        try:
            serv_sock.bind((self._host, self._port))
            serv_sock.listen()

            while True:
                conn, _ = serv_sock.accept()
                try:
                    self.serve_client(conn)
                except Exception as e:
                    print('Client serving failed', e)
        finally:
            serv_sock.close()

    def serve_client(self, conn):
        try:
            print("\nServing client")
            print("Parsing request")
            req = self.parse_request(conn)
            print("Handling request")
            resp = self.handle_request(req)
            print("Sending response")
            self.send_response(conn, resp)
```



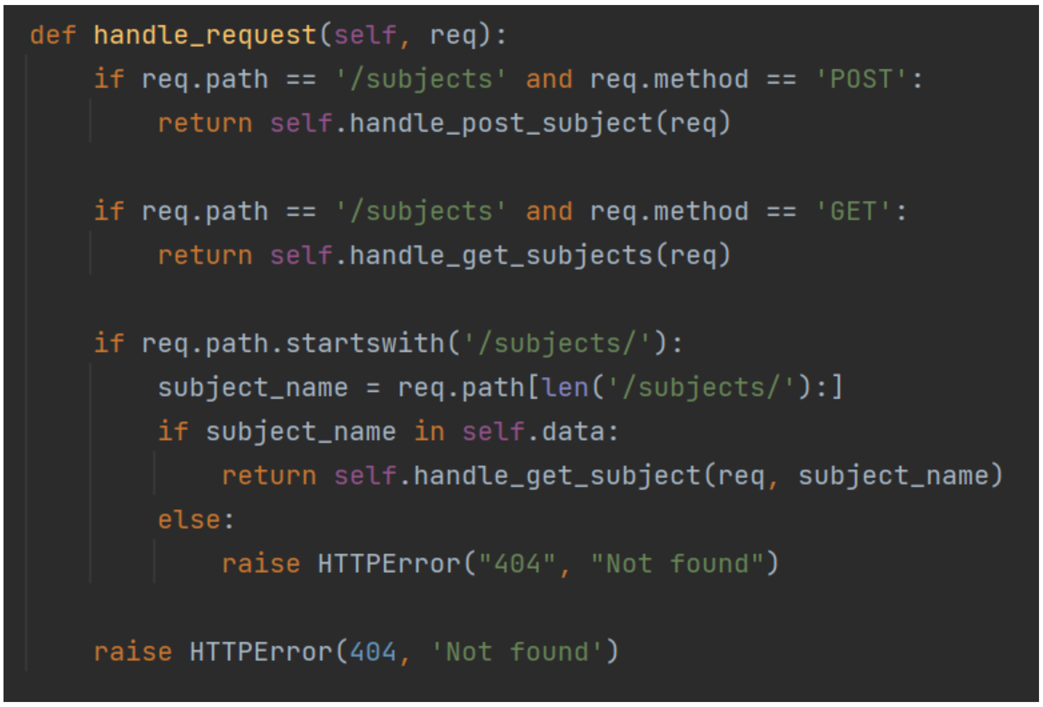
```python
        except ConnectionResetError:
            conn = None
        except Exception as e:
            self.send_error(conn, e)

        if conn:
            conn.close()

    def parse_request(self, conn):
        rfile = conn.makefile('rb')
        method, target, ver = self.parse_request_line(rfile)
        headers = self.parse_headers(rfile)
        host = headers.get('Host')
        if not host or host not in (self._server_name,
                                    f'{self._server_name}:{self._port}'):
            raise HTTPError(400, 'Bad request')
        return Request(method, target, ver, headers, rfile)
```



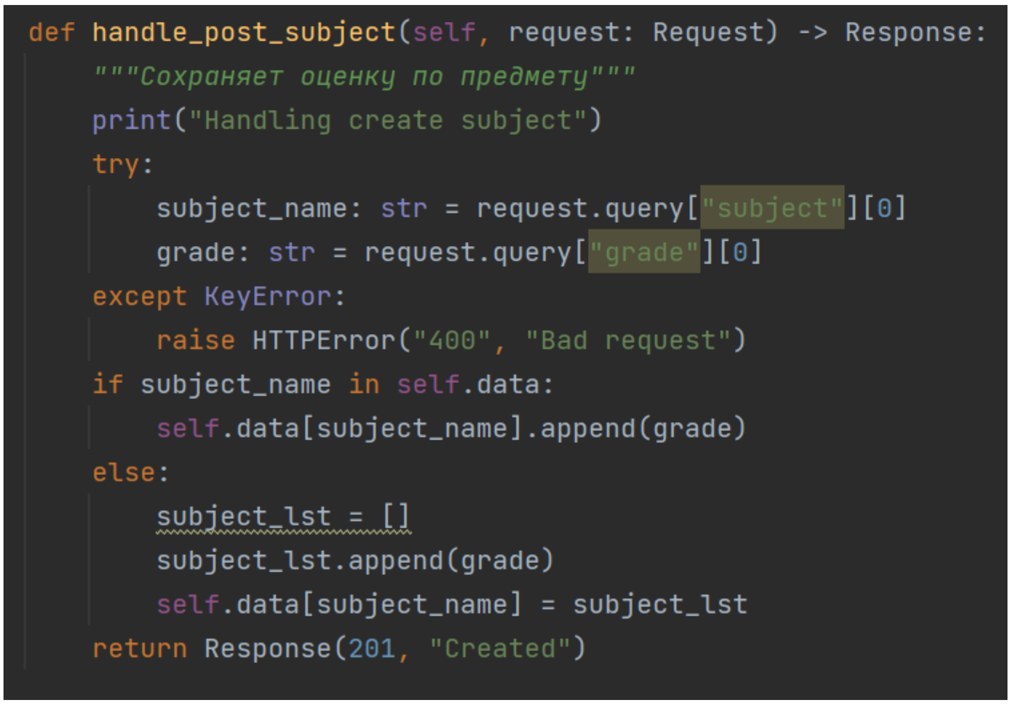
```python
def parse_headers(self, rfile):
    print("Parsing headers")
    headers = []
    while True:
        line = rfile.readline(MAX_LINE + 1)
        if len(line) > MAX_LINE:
            raise HTTPError(400, 'Header line is too long')

        if line in (b'\r\n', b'\n', b''):
            # завершаем чтение заголовков
            break

        headers.append(line)
        if len(headers) > MAX_HEADERS:
            raise HTTPError(400, 'Too many headers')
    sheaders = b''.join(headers).decode('iso-8859-1')
    return Parser().parsestr(sheaders)
```



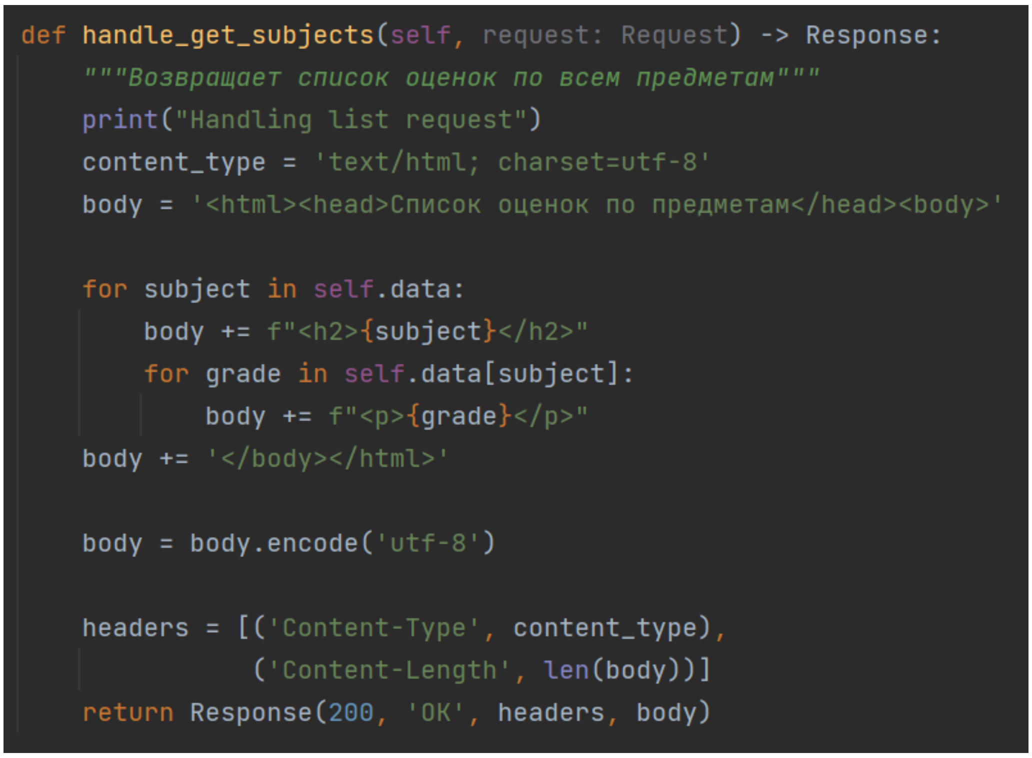
```python
def parse_request_line(self, rfile):
    print("Parsing request line")
    raw = rfile.readline(MAX_LINE + 1)
    if len(raw) > MAX_LINE:
        raise HTTPError(400, 'Request line is too long')

    req_line = str(raw, 'iso-8859-1')
    req_line = req_line.rstrip('\r\n')
    words = req_line.split()
    if len(words) != 3:
        raise HTTPError(400, 'Malformed request line')

    method, target, ver = words
    print(f"Target: {target}")
    if ver != 'HTTP/1.1':
        raise HTTPError(400, 'Unexpected HTTP version')
    return method, target, ver
```



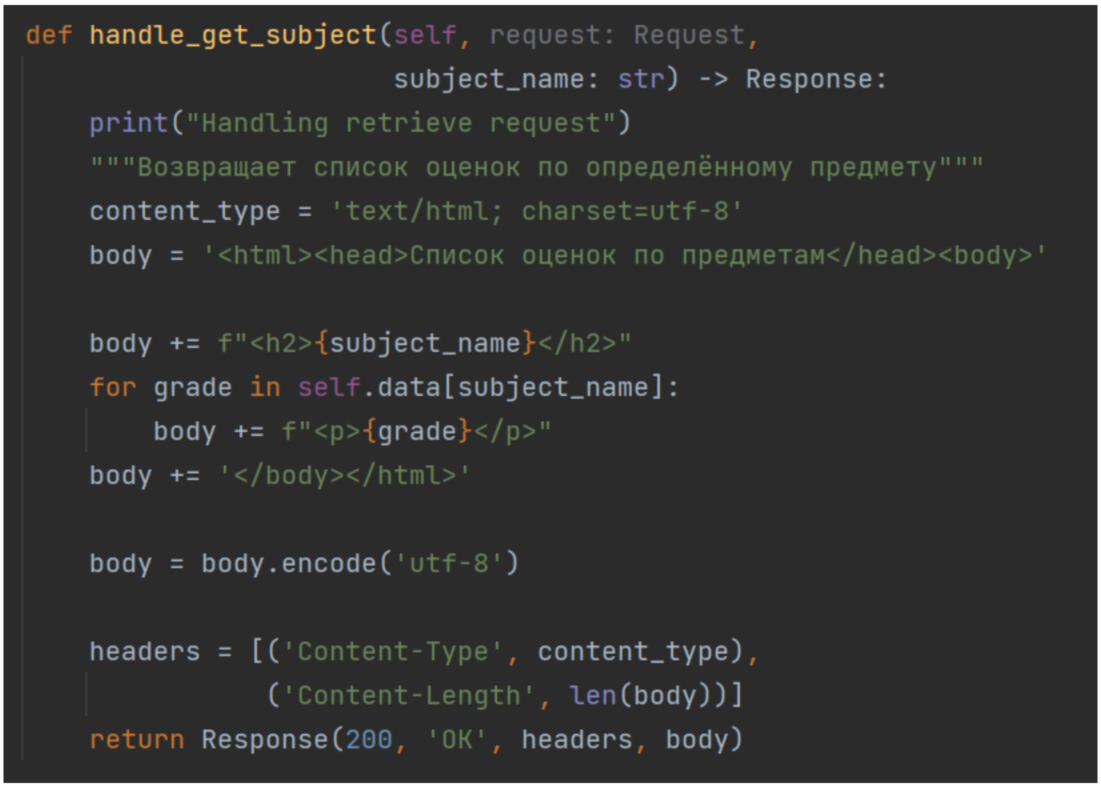
```
def handle_request(self, req):
    if req.path == '/subjects' and req.method == 'POST':
        return self.handle_post_subject(req)

    if req.path == '/subjects' and req.method == 'GET':
        return self.handle_get_subjects(req)

    if req.path.startswith('/subjects/'):
        subject_name = req.path[len('/subjects/'):]
        if subject_name in self.data:
            return self.handle_get_subject(req, subject_name)
        else:
            raise HTTPError("404", "Not found")

    raise HTTPError(404, 'Not found')
```



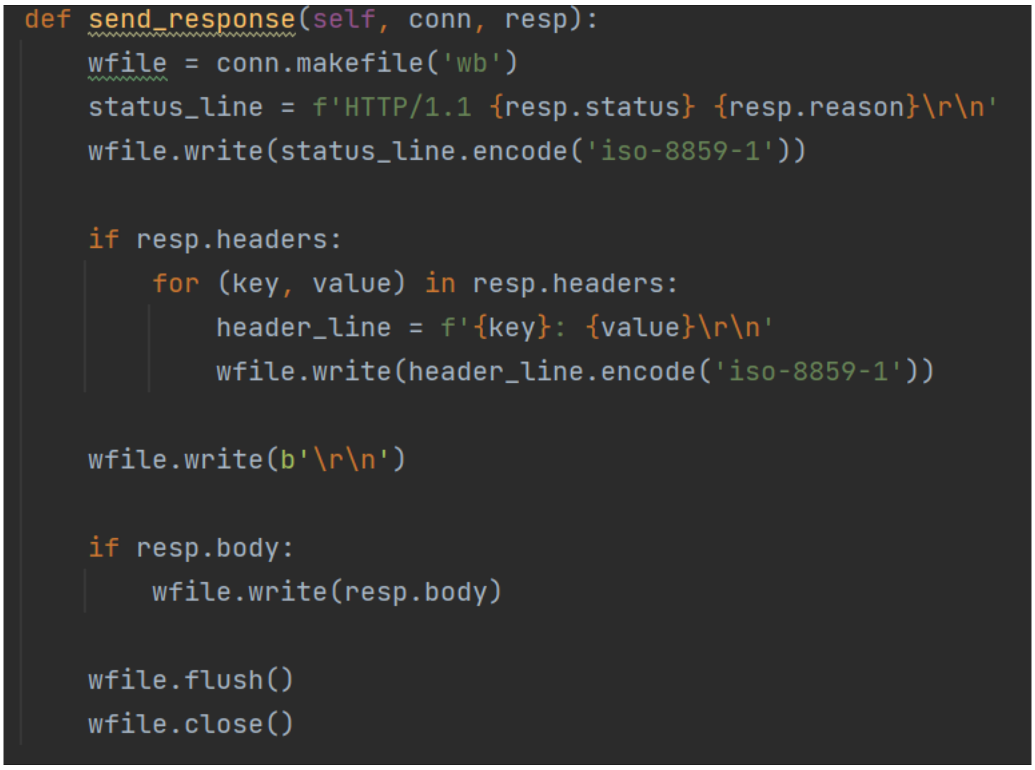
```
def handle_post_subject(self, request: Request) -> Response:
    """Сохраняет оценку по предмету"""
    print("Handling create subject")
    try:
        subject_name: str = request.query["subject"][0]
        grade: str = request.query["grade"][0]
    except KeyError:
        raise HTTPError("400", "Bad request")
    if subject_name in self.data:
        self.data[subject_name].append(grade)
    else:
        subject_lst = []
        subject_lst.append(grade)
        self.data[subject_name] = subject_lst
    return Response(201, "Created")
```



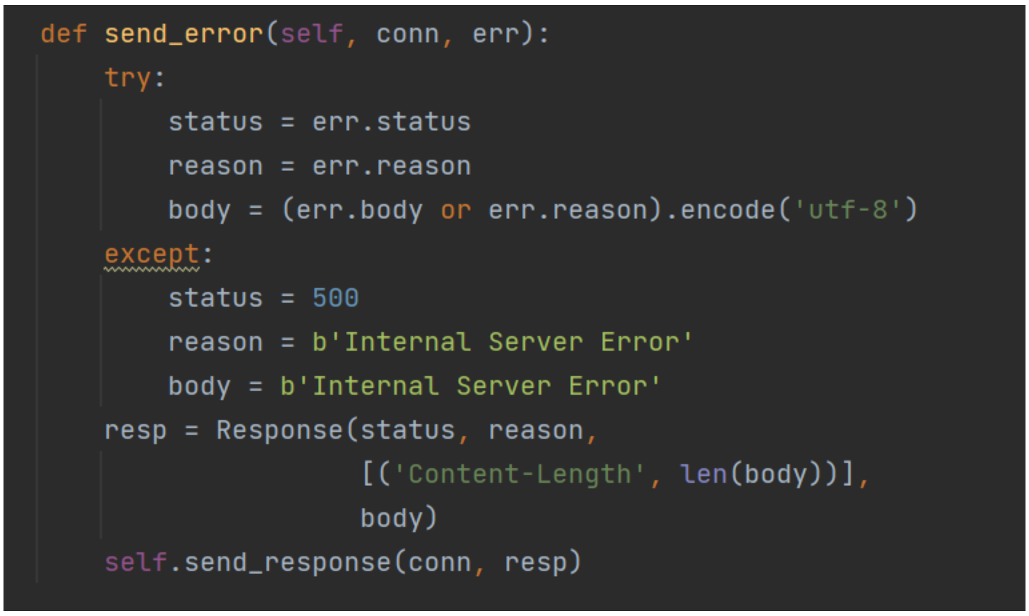
```python
def handle_get_subjects(self, request: Request) -> Response:
    """Возвращает список оценок по всем предметам"""
    print("Handling list request")
    content_type = 'text/html; charset=utf-8'
    body = '<html><head>Список оценок по предметам</head><body>'

    for subject in self.data:
        body += f"<h2>{subject}</h2>"
        for grade in self.data[subject]:
            body += f"<p>{grade}</p>"
    body += '</body></html>'

    body = body.encode('utf-8')

    headers = [('Content-Type', content_type),
               ('Content-Length', len(body))]
    return Response(200, 'OK', headers, body)
```



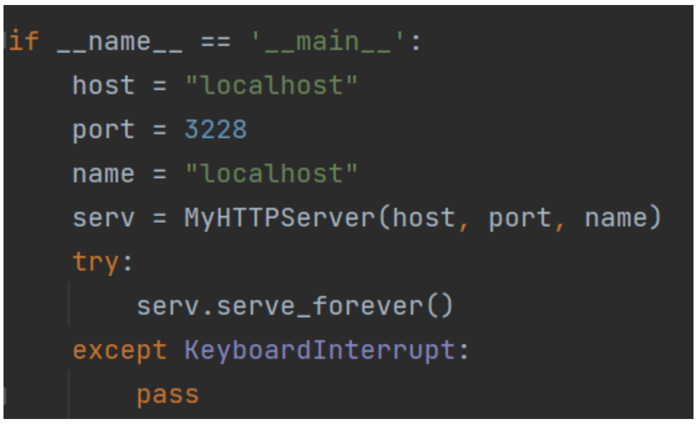
```python
def handle_get_subject(self, request: Request,
                       subject_name: str) -> Response:
    print("Handling retrieve request")
    """Возвращает список оценок по определённому предмету"""
    content_type = 'text/html; charset=utf-8'
    body = '<html><head>Список оценок по предметам</head><body>'

    body += f"<h2>{subject_name}</h2>"
    for grade in self.data[subject_name]:
        body += f"<p>{grade}</p>"
    body += '</body></html>'

    body = body.encode('utf-8')

    headers = [('Content-Type', content_type),
               ('Content-Length', len(body))]
    return Response(200, 'OK', headers, body)
```



```
def send_response(self, conn, resp):
    wfile = conn.makefile('wb')
    status_line = f'HTTP/1.1 {resp.status} {resp.reason}\r\n'
    wfile.write(status_line.encode('iso-8859-1'))

    if resp.headers:
        for (key, value) in resp.headers:
            header_line = f'{key}: {value}\r\n'
            wfile.write(header_line.encode('iso-8859-1'))

    wfile.write(b'\r\n')

    if resp.body:
        wfile.write(resp.body)

    wfile.flush()
    wfile.close()
```

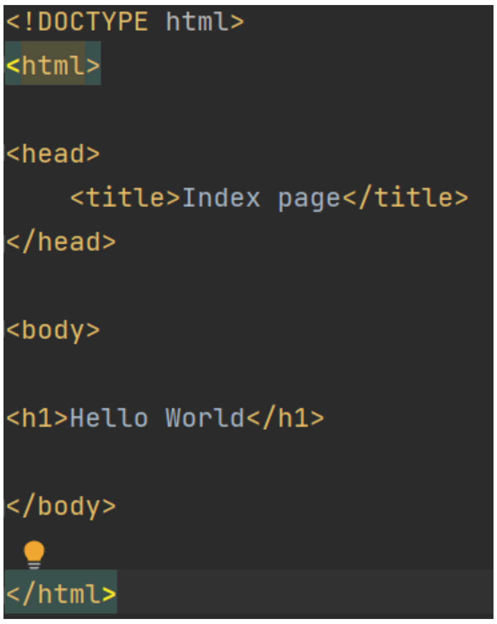


```
def send_error(self, conn, err):
    try:
        status = err.status
        reason = err.reason
        body = (err.body or err.reason).encode('utf-8')
    except:
        status = 500
        reason = b'Internal Server Error'
        body = b'Internal Server Error'
    resp = Response(status, reason,
                    [('Content-Length', len(body))],
                    body)
    self.send_response(conn, resp)
```



```
if __name__ == '__main__':
    host = "localhost"
    port = 3228
    name = "localhost"
    serv = MyHTTPServer(host, port, name)
    try:
        serv.serve_forever()
    except KeyboardInterrupt:
        pass
```

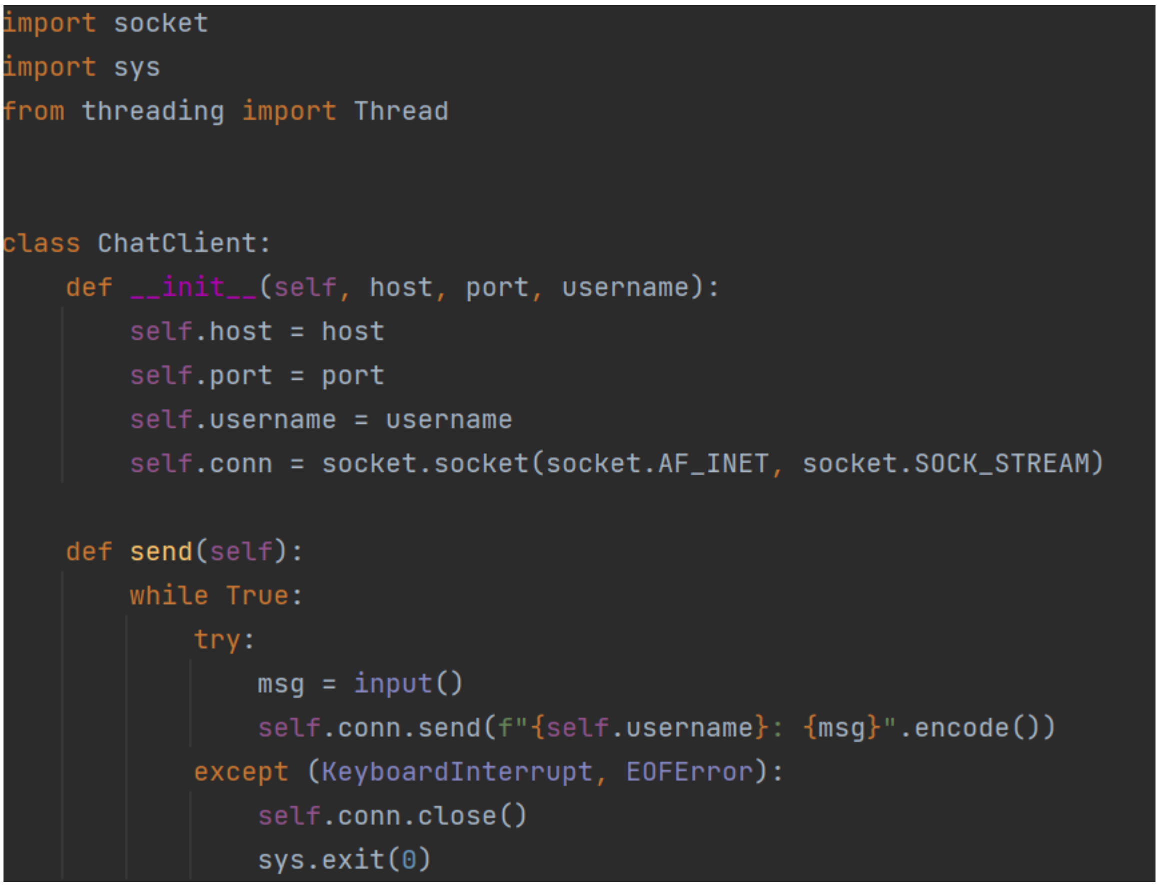
Index.html:



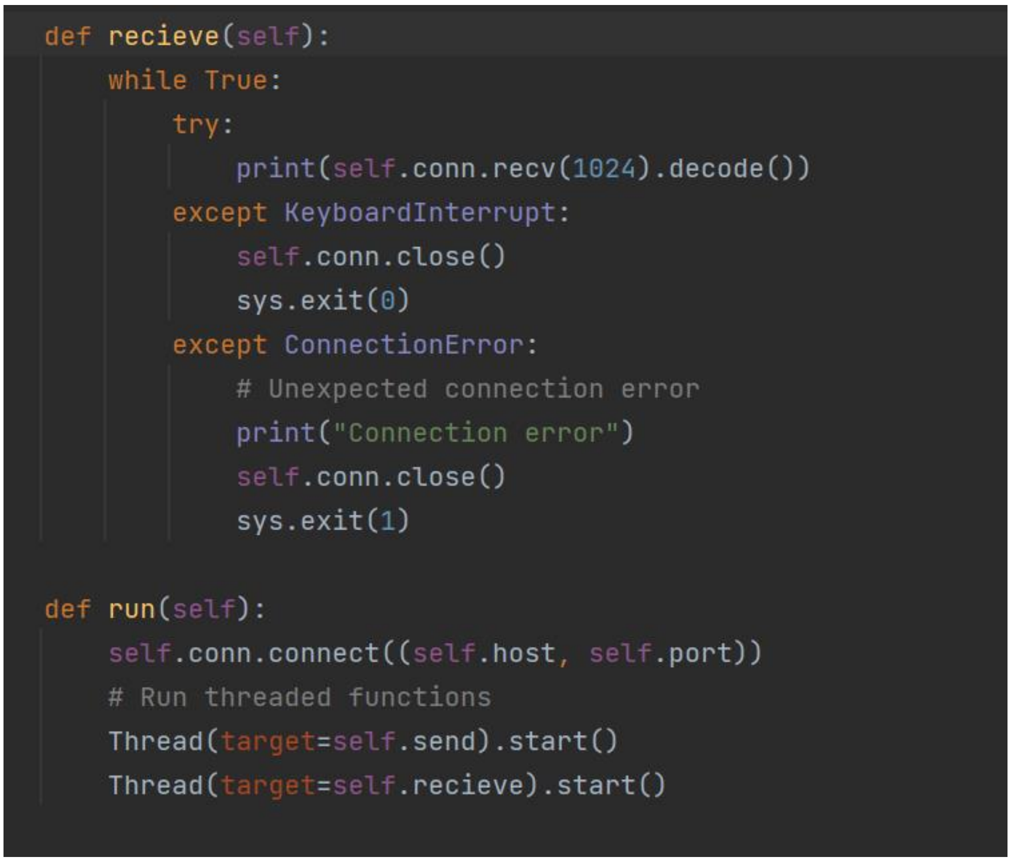
```html
<!DOCTYPE html>
<html>

<head>
    <title>Index page</title>
</head>

<body>

<h1>Hello World</h1>

</body>

</html>
```

Задание 4:

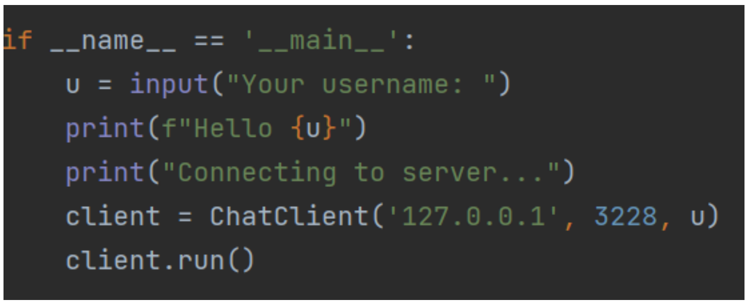
Клиент:



```python
import socket
import sys
from threading import Thread


class ChatClient:
    def __init__(self, host, port, username):
        self.host = host
        self.port = port
        self.username = username
        self.conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def send(self):
        while True:
            try:
                msg = input()
                self.conn.send(f"{self.username}: {msg}".encode())
            except (KeyboardInterrupt, EOFError):
                self.conn.close()
                sys.exit(0)
```

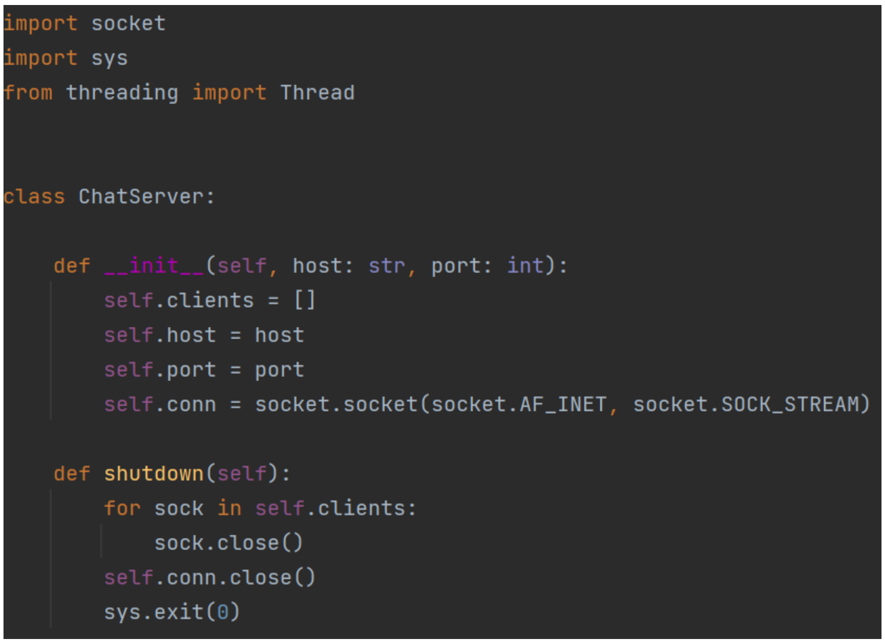


```python
    def recieve(self):
        while True:
            try:
                print(self.conn.recv(1024).decode())
            except KeyboardInterrupt:
                self.conn.close()
                sys.exit(0)
            except ConnectionError:
                # Unexpected connection error
                print("Connection error")
                self.conn.close()
                sys.exit(1)

    def run(self):
        self.conn.connect((self.host, self.port))
        # Run threaded functions
        Thread(target=self.send).start()
        Thread(target=self.recieve).start()
```

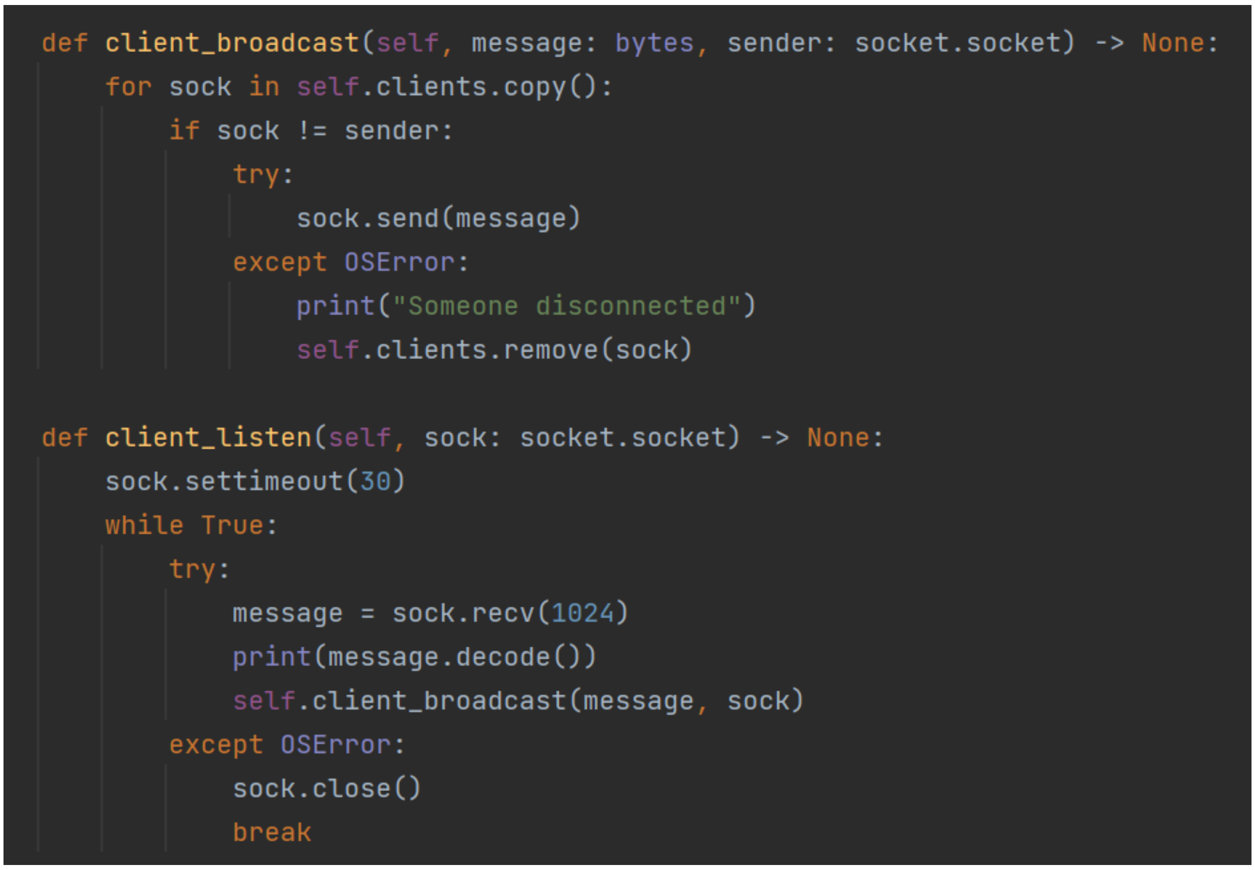


```python
if __name__ == '__main__':
    u = input("Your username: ")
    print(f"Hello {u}")
    print("Connecting to server...")
    client = ChatClient('127.0.0.1', 3228, u)
    client.run()
```

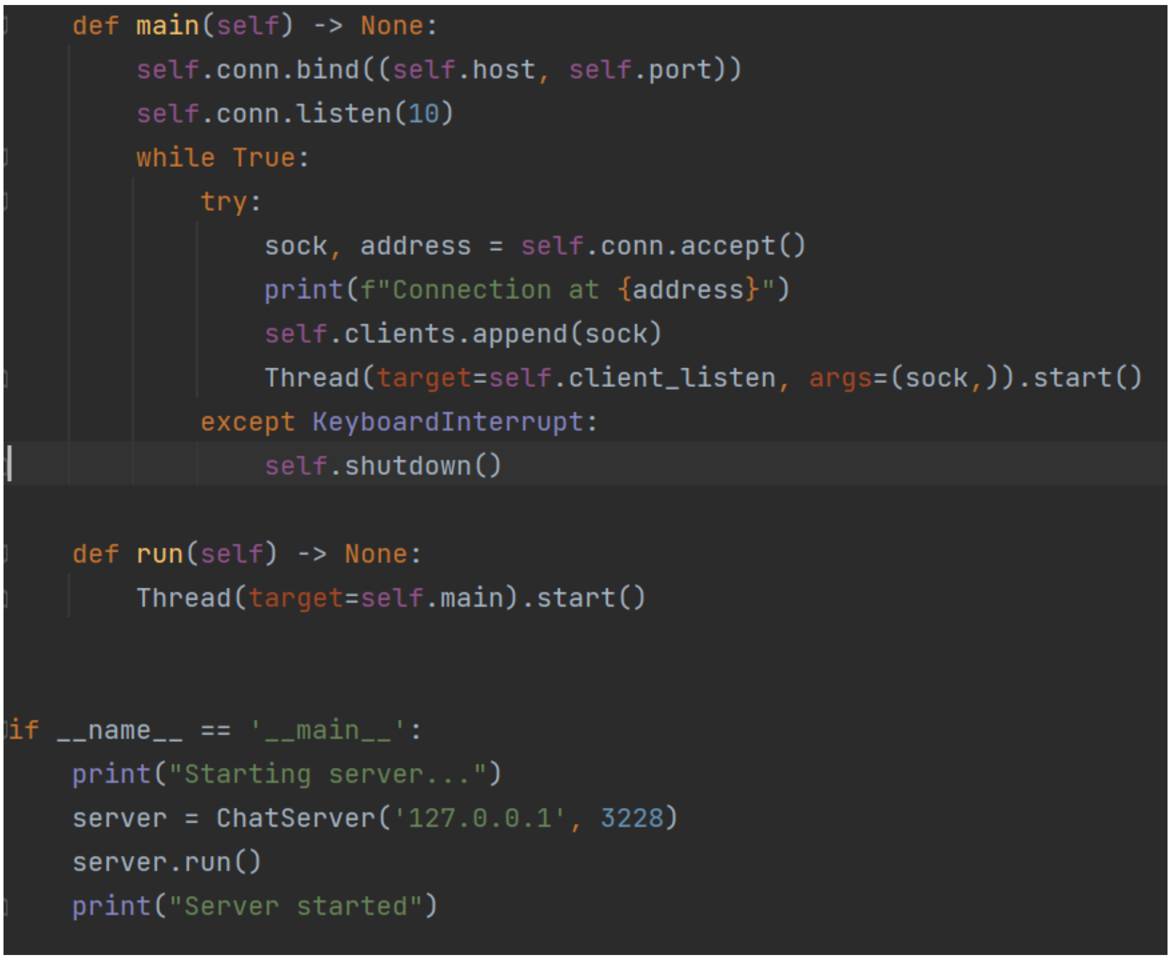
Сервер:



```python
import socket
import sys
from threading import Thread


class ChatServer:

    def __init__(self, host: str, port: int):
        self.clients = []
        self.host = host
        self.port = port
        self.conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def shutdown(self):
        for sock in self.clients:
            sock.close()
        self.conn.close()
        sys.exit(0)
```



```python
def client_broadcast(self, message: bytes, sender: socket.socket) -> None:
    for sock in self.clients.copy():
        if sock != sender:
            try:
                sock.send(message)
            except OSError:
                print("Someone disconnected")
                self.clients.remove(sock)

def client_listen(self, sock: socket.socket) -> None:
    sock.settimeout(30)
    while True:
        try:
            message = sock.recv(1024)
            print(message.decode())
            self.client_broadcast(message, sock)
        except OSError:
            sock.close()
            break
```



```python
    def main(self) -> None:
        self.conn.bind((self.host, self.port))
        self.conn.listen(10)
        while True:
            try:
                sock, address = self.conn.accept()
                print(f"Connection at {address}")
                self.clients.append(sock)
                Thread(target=self.client_listen, args=(sock,)).start()
            except KeyboardInterrupt:
                self.shutdown()

    def run(self) -> None:
        Thread(target=self.main).start()


if __name__ == '__main__':
    print("Starting server...")
    server = ChatServer('127.0.0.1', 3228)
    server.run()
    print("Server started")
```

Вывод:

В ходе работы были написаны 4 программы на Python,

использующие сокеты для коммуникации между клиентом и сервером.