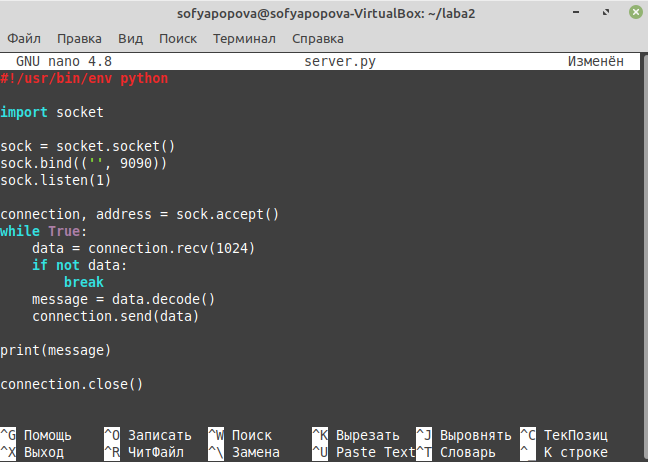
**Задания для выполнения**

Изначально мои клиент и сервер выглядели так:

1. Создать простой TCP-сервер, который принимает от клиента строку (порциями по 1 КБ) и возвращает ее. (Эхо-сервер).



1. Сервер должен выводить в консоль служебные сообщения (с пояснениями) при наступлении любых событий:

* Запуск сервера;



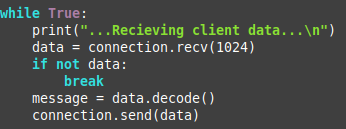
* Начало прослушивания порта;



* Подключение клиента;



* Прием данных от клиента;



* Отправка данных клиенту;



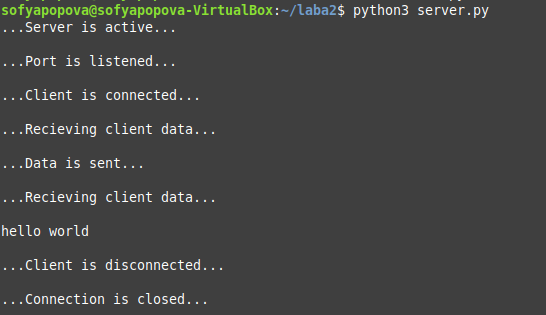
* Отключение клиента;



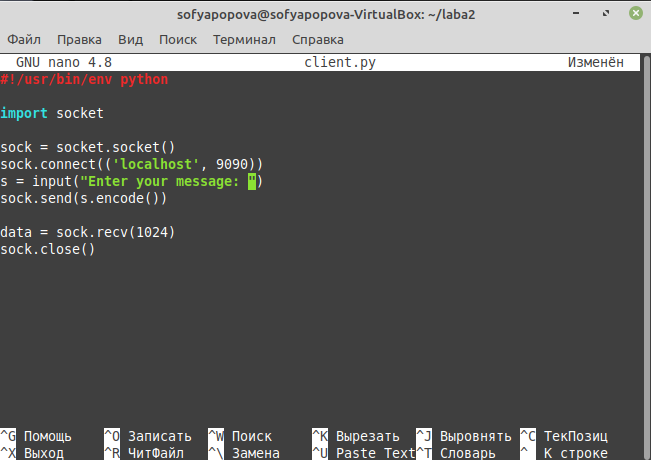
* Остановка сервера.



Итог:



1. Напишите простой TCP-клиент, который устанавливает соединение с сервером, считывает строку со стандартного ввода и посылает его серверу.



1. Клиент должен выводить в консоль служебные сообщения (с пояснениями) при наступлении любых событий:

* Соединение с сервером;



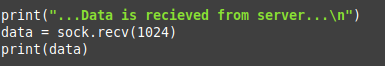
* Разрыв соединения с сервером;



* Отправка данных серверу;



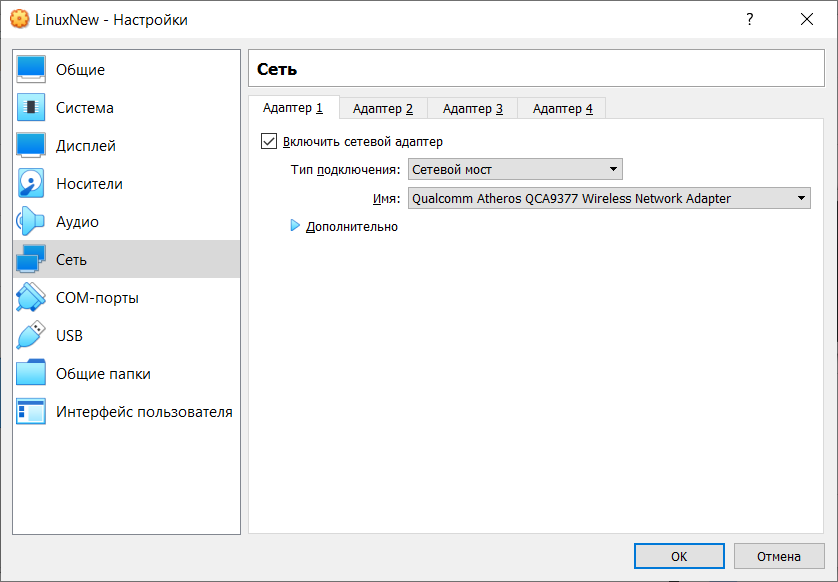
* Прием данных от сервера.

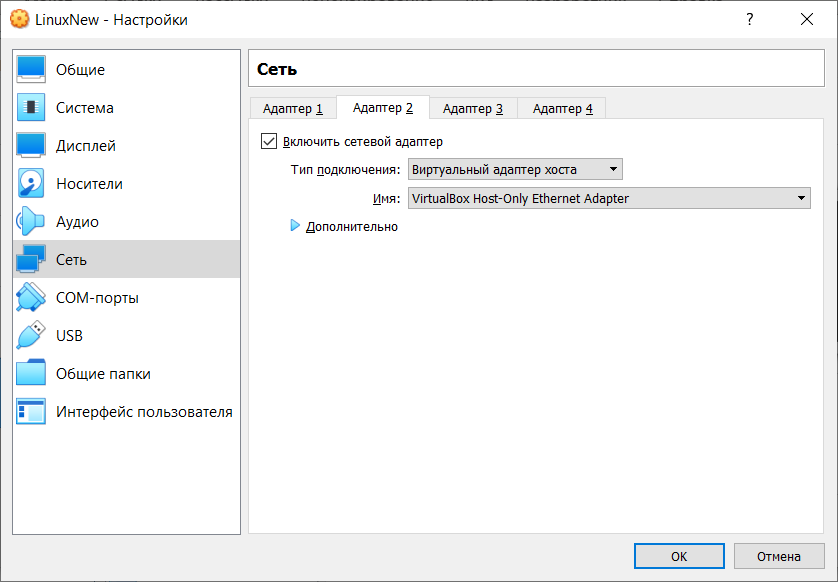


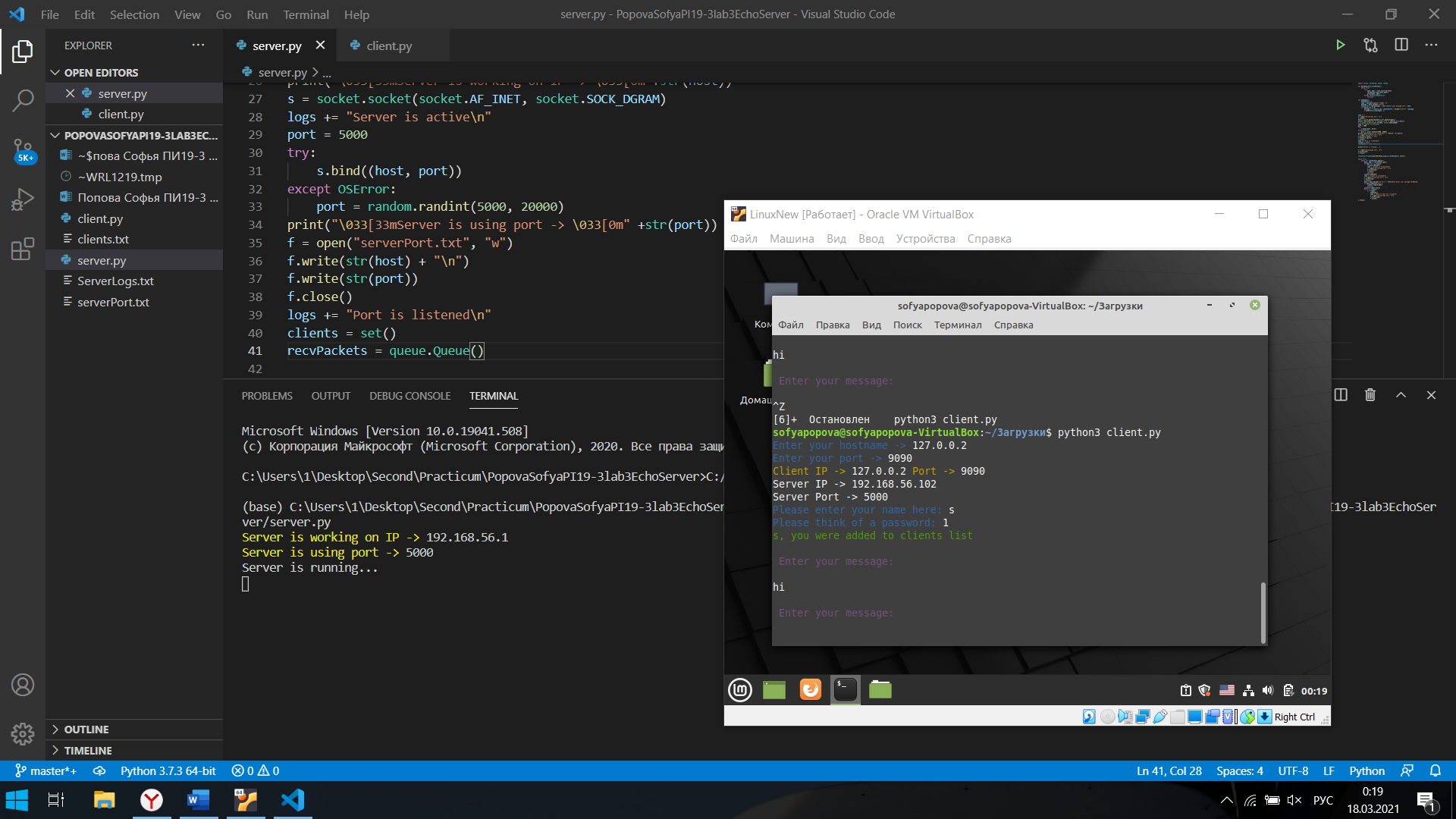
Дальше сложно скринить для отчёта каждый шаг по мере выполнения, потому что код может дополняться, модифицироваться и т.д., поэтому я вставляю скрины уже после выполнения всей работы.

**Задания для самостоятельного выполнения**

1. Проверьте возможность подключения к серверу с локальной, виртуальной и удаленной машины.

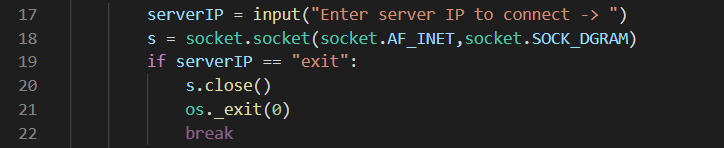




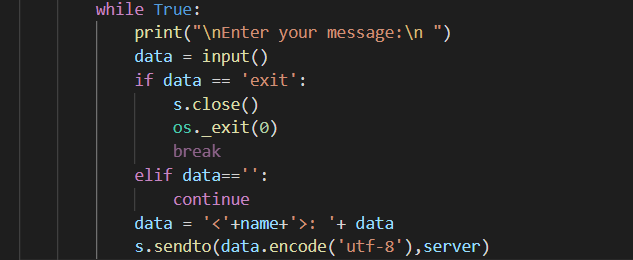


1. Модифицируйте код клиента таким образом, чтобы он читал строки в цикле до тех пор, пока клиент не введет “exit”. Можно считать, что это команда разрыва соединения со стороны клиента.

Если клиент захотел выйти ещё до того, как подключился к серверу:

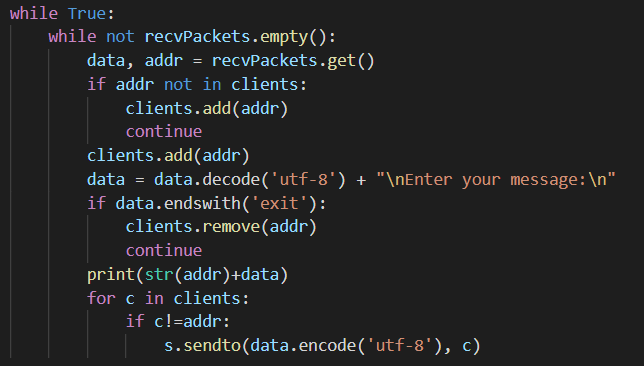


Если клиент уже подключился, но теперь хочет отключиться:



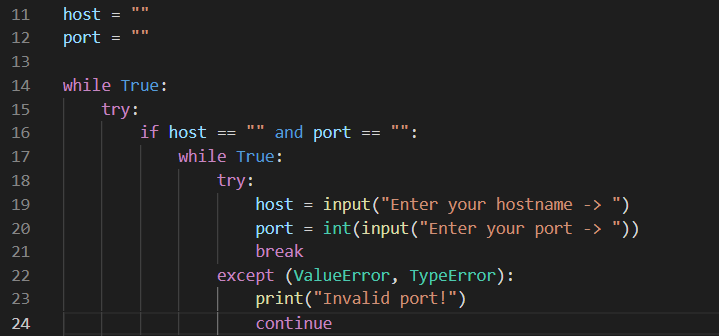
1. Модифицируйте код сервера таким образом, чтобы при разрыве соединения клиентом он продолжал слушать данный порт и, таким образом, был доступен для повторного подключения.

Это достигается благодаря бесконечному циклу:



1. Модифицируйте код клиента и сервера таким образом, чтобы номер порта и имя хоста (для клиента) они спрашивали у пользователя. Реализовать безопасный ввод данных и значения по умолчанию.

Безопасный ввод данных:

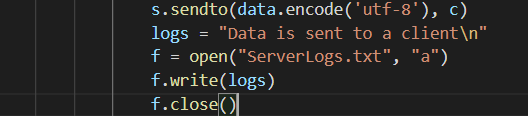


1. Модифицировать код сервера таким образом, чтобы все служебные сообщения выводились не в консоль, а в специальный лог-файл.

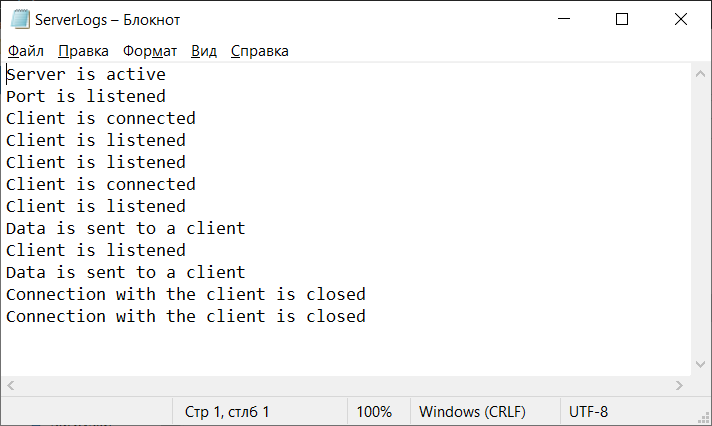
В начале программы создаю файл (или открываю на перезапись):



И по всему коду вот такие вот дозаписи событий в файл:

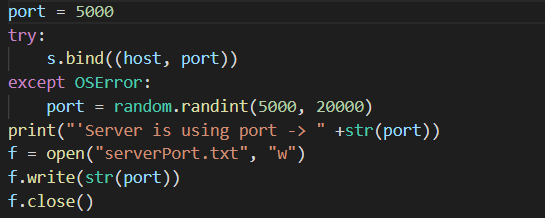


В итоге после работы программы это выглядит как-то так:

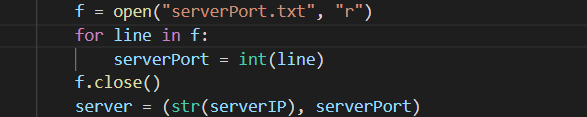


1. Модифицируйте код сервера таким образом, чтобы он автоматически изменял номер порта, если он уже занят. Сервер должен выводить в консоль номер порта, который он слушает.

Вот так серверу назначается порт:



Он записывается в файл, чтобы клиент мог его оттуда прочитать и соединиться:

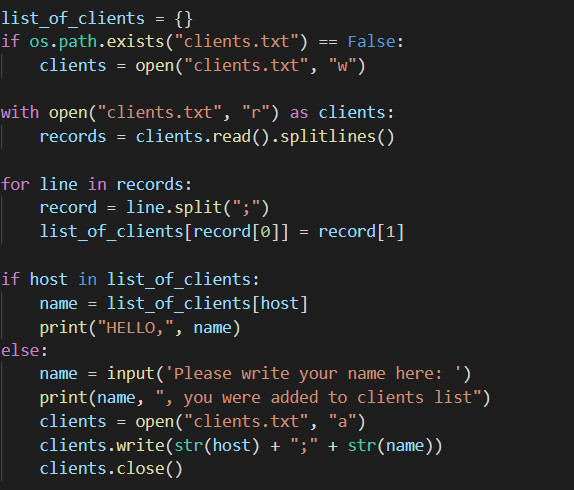


Также номер порта выводится в консоль:

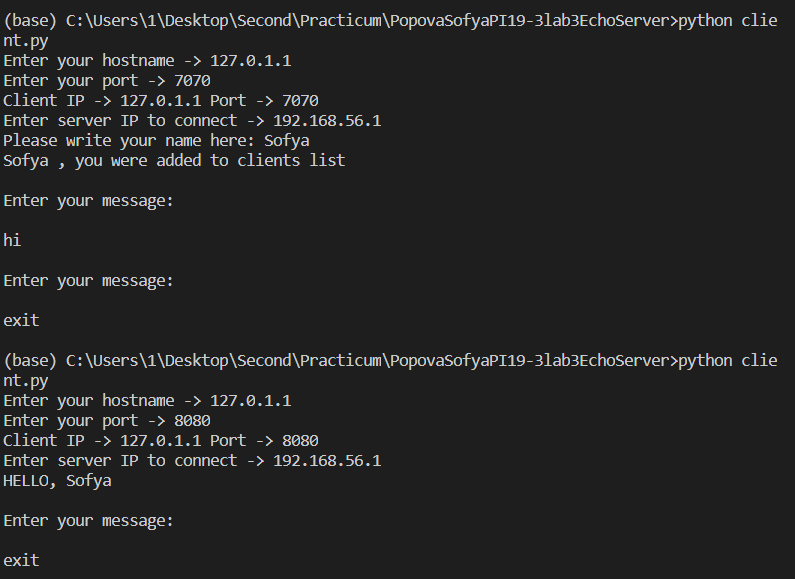


1. Реализовать сервер идентификации. Сервер должен принимать соединения от клиента и проверять, известен ли ему уже этот клиент (по IP-адресу). Если известен, то поприветствовать его по имени. Если неизвестен, то запросить у пользователя имя и записать его в файл. Файл хранить в произвольном формате.

Код:



В итоге сначала нас ещё не знают, а уже при втором запуске узнают и приветствуют:

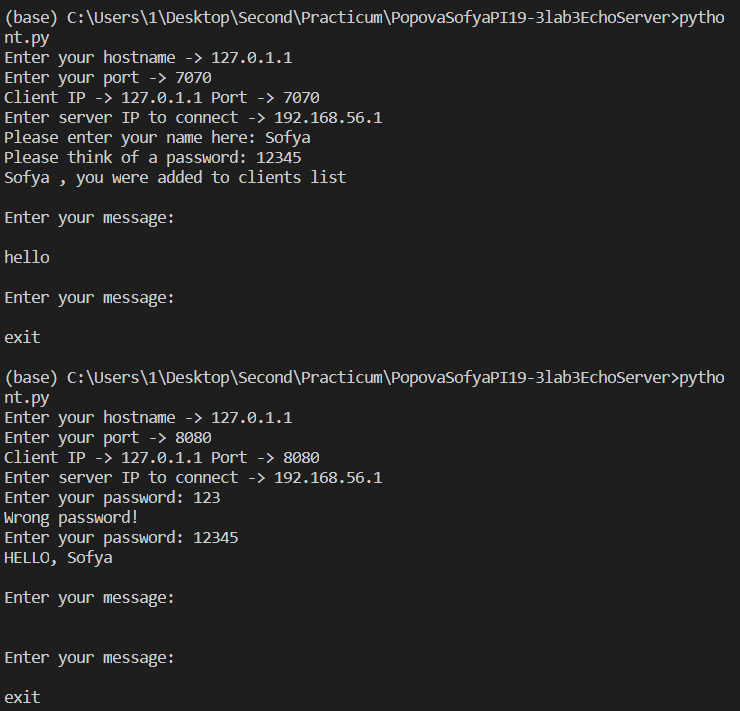
****

1. Реализовать сервер аутентификации. Похоже на предыдущее задание, но вместе с именем пользователя сервер отслеживает и проверяет пароли. Дополнительные баллы за безопасное хранение паролей. Дополнительные баллы за поддержание сессии на основе токена наподобие cookies

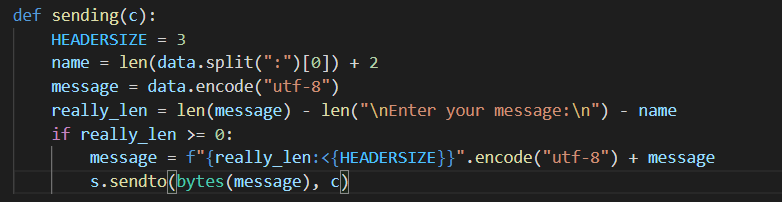
Немножко поменяла код:



Теперь при первом запуске просит регистрацию, а при последующих – авторизацию:



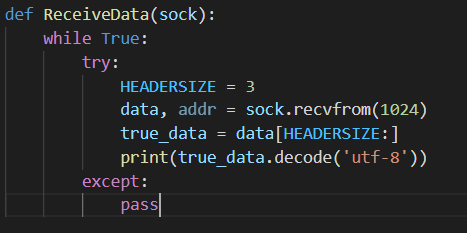
1. Напишите вспомогательные функции, которые реализуют отправку и принятие текстовых сообщений в сокет. Функция отправки должна дополнять сообщение заголовком фиксированной длины, в котором содержится информация о длине сообщения. Функция принятия должна читать сообщение с учетом заголовка. В дополнении реализуйте преобразование строки в байтовый массив и обратно в этих же функциях. Дополнително оценивается, если эти функции будут реализованы как унаследованное расширение класса socket библиотеки socket.



Так это работало:



Но это некрасиво, так что:

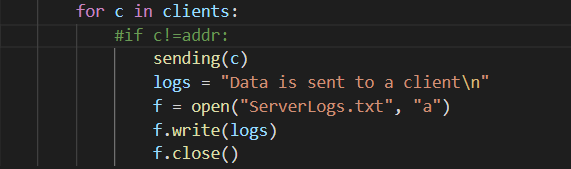


Эти заголовки можно и не выводить:



1. Дополните код клиента и сервера таким образом, чтобы они могли посылать друг другу множественные сообщения один в ответ на другое.

Чтобы сообщение посылалось и самому клиенту, отправившему сообщения, нужно сделать так:



Тогда:



Но так некрасиво, так что раскомментирую обратно.

1. Напишите многопользовательский чат.

Работа моей итоговой программы выглядит так:

