

Университет ИТМО

Факультет «Инфокоммуникационных технологий»
Направление подготовки «название направления подготовки»

Лабораторная работа №6
«Сокеты»

Выполнил:
Кузнецов Никита Сергеевич
Группа К33212
Проверил:
Марченко Елена Вадимовна

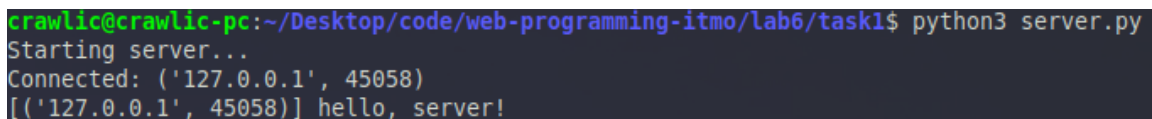
Санкт-Петербург
2023

Цель работы: изучить работу и устройство сокетов в Python, разработать различные приложения с клиентской и серверной и частью, в том числе многопользовательский чат и получение HTML-страницы через сокет.

Ход работы:

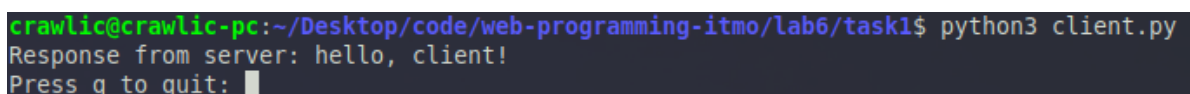
1 задание: реализовать клиентскую и серверную часть приложения. Клиент отправляет серверу сообщение «Hello, server». Сообщение должно отразиться на стороне сервера. Сервер в ответ отправляет клиенту сообщение «Hello, client». Сообщение должно отобразиться у клиента.

Для выполнения данного задания были реализованы два Python файла для клиентской и серверной части (приложения 1.1-1.2). В серверной части создавался сам сервер, а также принимались любые соединения, а каждый новый адрес выводился в консоль. В клиентской части был реализован код подключения к серверу, а также отправка сообщения “hello, server!” (рис. 1). В ответ на это сообщение клиенту приходило сообщение от сервера “hello, client!” (рис. 2), и данный ответ отражался в консоли.



```
crawlic@crawlic-pc:~/Desktop/code/web-programming-itmo/lab6/task1$ python3 server.py
Starting server...
Connected: ('127.0.0.1', 45058)
[('127.0.0.1', 45058)] hello, server!
```

Рисунок 1 - Подключение к серверу клиентом и отправка сообщения “hello, server”.



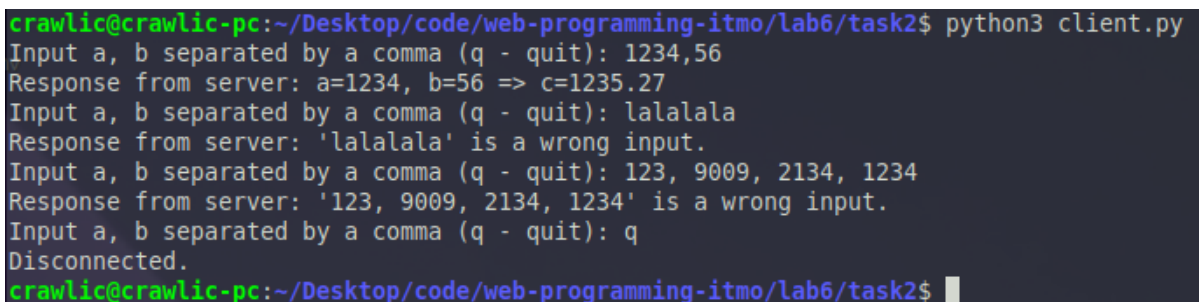
```
crawlic@crawlic-pc:~/Desktop/code/web-programming-itmo/lab6/task1$ python3 client.py
Response from server: hello, client!
Press q to quit: █
```

Рисунок 2 - Получение ответа от сервера на клиентской стороне.

2 задание: реализовать клиентскую и серверную часть приложения. Клиент запрашивает у сервера выполнение математической операции, параметры,

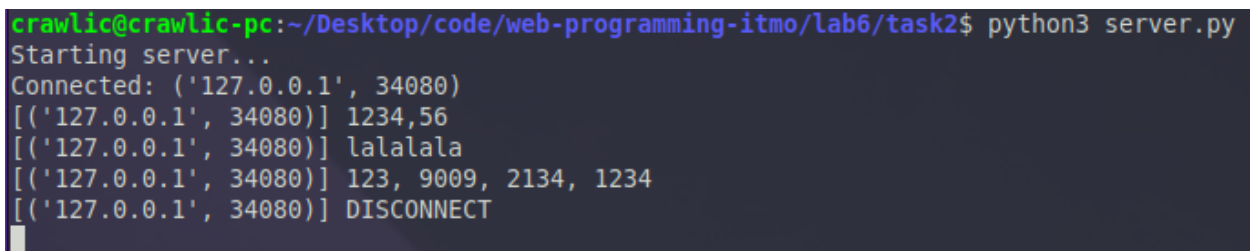
которые вводятся с клавиатуры. Сервер обрабатывает полученные данные и возвращает результат клиенту.

Исходя из варианта, была выбрана теорема Пифагора. Для реализации данного задания были созданы аналогично файлы с клиентской и серверной частью (приложения 2.1-2.2). В серверной части были реализованы функции для валидации ввода двух чисел через запятую, а также подсчет теоремы Пифагора для a и b . В клиентской части предлагалось ввести два числа, после чего приходил ответ либо в виде рационального числа, либо сообщение о неправильном вводе (рис. 3). Также, был реализован функционал выхода пользователя с сервера при отправке сообщения о выходе.



```
crawlic@crawlic-pc:~/Desktop/code/web-programming-itmo/lab6/task2$ python3 client.py
Input a, b separated by a comma (q - quit): 1234,56
Response from server: a=1234, b=56 => c=1235.27
Input a, b separated by a comma (q - quit): lalalala
Response from server: 'lalalala' is a wrong input.
Input a, b separated by a comma (q - quit): 123, 9009, 2134, 1234
Response from server: '123, 9009, 2134, 1234' is a wrong input.
Input a, b separated by a comma (q - quit): q
Disconnected.
crawlic@crawlic-pc:~/Desktop/code/web-programming-itmo/lab6/task2$
```

Рисунок 3 - Реализация подсчета теоремы Пифагора для клиента при вводе двух чисел.



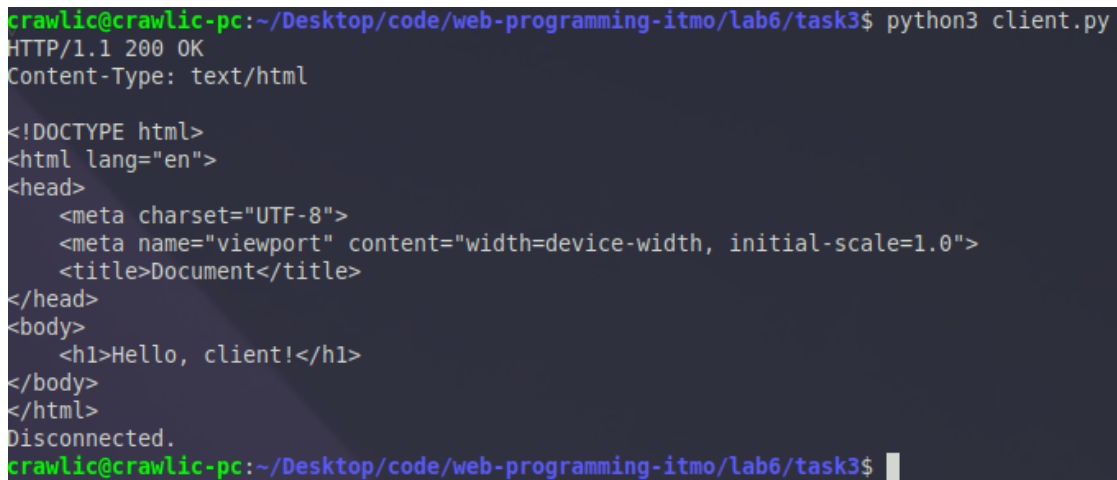
```
crawlic@crawlic-pc:~/Desktop/code/web-programming-itmo/lab6/task2$ python3 server.py
Starting server...
Connected: ('127.0.0.1', 34080)
[('127.0.0.1', 34080)] 1234,56
[('127.0.0.1', 34080)] lalalala
[('127.0.0.1', 34080)] 123, 9009, 2134, 1234
[('127.0.0.1', 34080)] DISCONNECT
```

Рисунок 4 - Сообщения от клиента в консоли сервера.

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

Для выполнения данного задания была реализована клиентская и серверная часть (приложения 3.1-3.2), а также HTML файл для ответа. При запросе на сервер

клиенту отправлялся HTTP ответ со статусом 200 OK и заголовком `ContentType: text/html`, а также само содержимое HTML файла (рис. 5).



```
crawlic@crawlic-pc:~/Desktop/code/web-programming-itmo/lab6/task3$ python3 client.py
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Hello, client!</h1>
</body>
</html>
Disconnected.
crawlic@crawlic-pc:~/Desktop/code/web-programming-itmo/lab6/task3$
```

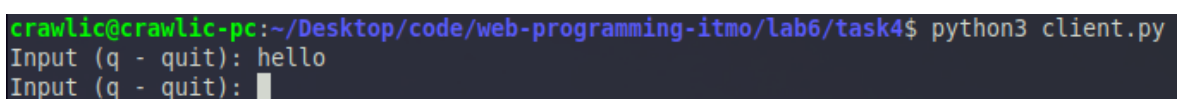
Рисунок 5 - HTTP ответ HTML страницей клиенту от сервера.

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

Для выполнения этого задания было принято решение разделить его на этапы:

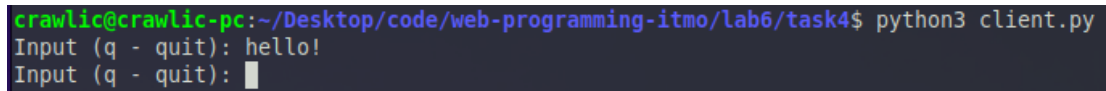
- 1) Создание клиентской и серверной части (приложения 4.1-4.2).
- 2) Создание “особого” клиента для общего чата (приложение 4.3).

В клиентской части было выполнено базовое отправление сообщений. Затем, на сервере эти сообщения принимались и отправлялись в общий “чат”, который находился в постоянном режиме ожидания новых сообщений. Таким образом, клиенты могли общаться друг с другом, видя все сообщения (как свои так и чужие) в общем чате в реальном времени. Демонстрация данного процесса приведена на рисунках 6-8.



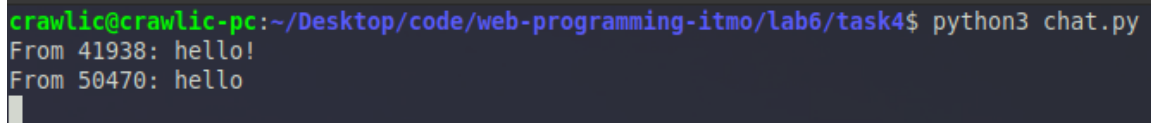
```
crawlic@crawlic-pc:~/Desktop/code/web-programming-itmo/lab6/task4$ python3 client.py
Input (q - quit): hello
Input (q - quit):
```

Рисунок 6 - Отправка сообщения в чат от первого клиента.

A terminal window with a dark background. The prompt is 'crawlic@crawlic-pc:~/Desktop/code/web-programming-itmo/lab6/task4\$'. The user has entered 'python3 client.py'. The program output shows 'Input (q - quit): hello!' followed by 'Input (q - quit):' and a cursor.

```
crawlic@crawlic-pc:~/Desktop/code/web-programming-itmo/lab6/task4$ python3 client.py
Input (q - quit): hello!
Input (q - quit):
```

Рисунок 7 - Отправка сообщения в чат со второго клиента.

A terminal window with a dark background. The prompt is 'crawlic@crawlic-pc:~/Desktop/code/web-programming-itmo/lab6/task4\$'. The user has entered 'python3 chat.py'. The program output shows 'From 41938: hello!' followed by 'From 50470: hello' and a cursor.

```
crawlic@crawlic-pc:~/Desktop/code/web-programming-itmo/lab6/task4$ python3 chat.py
From 41938: hello!
From 50470: hello
```

Рисунок 8 - Оба сообщения клиентов появились в общем многопользовательском чате.

Вывод:

В результате лабораторной работы были исследованы и изучены возможности сокетов на языке Python. Были реализованы различные варианты общения между клиентской и серверной частью, а также общение между клиентами в многопользовательском чате.

Приложение 1.1. Серверная часть для задания 1.

```
import threading
import socket

ADDR = ("localhost", 5050)
FORMAT = "utf-8"
MAX_SIZE = 1024

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

def handle_client(conn: socket.socket, addr: tuple[str, int]):
    print(f"Connected: {addr}")

    connected = True
    while connected:
        msg = conn.recv(MAX_SIZE).decode(FORMAT)
        print(f"[{addr}] {msg}")
        conn.send("hello, client!".encode(FORMAT))

    conn.close()

def start():
    print("Starting server...")
    server.bind(ADDR)
    server.listen()
    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handle_client, args=(conn, addr))
        thread.start()

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

Приложение 1.2. Клиентская часть для задания 1.

```
import socket

from server import ADDR, FORMAT, MAX_SIZE

def send(client: socket.socket, msg: str):
    message = msg.encode(FORMAT)
    client.send(message)

def start():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(ADDR)
```

```
send(client, "hello, server!")
while True:
    response = client.recv(MAX_SIZE).decode(FORMAT)
    print(f"Response from server: {response}")
    q = input("Press q to quit: ")
    if q == "q":
        break

client.close()

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

Приложение 2.1. Серверная часть для задания 2.

```
import math
import threading
import socket

ADDR = ("localhost", 5050)
FORMAT = "utf-8"
DISCONNECT_MESSAGE = "DISCONNECT"
MAX_SIZE = 1024

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

def pythagorean_theorem(a: int, b: int):
    return math.sqrt(a**2 + b**2)

def get_input(msg: str) -> tuple[int, int]:
    a, b = (int(item) for item in msg.split(","))
    return a, b

def handle_client(conn: socket.socket, addr: tuple[str, int]):
    print(f"Connected: {addr}")

    connected = True
    while connected:
        msg = conn.recv(MAX_SIZE).decode(FORMAT)
        if not msg:
            break
        try:
            a, b = get_input(msg)
            conn.send(
                f"{a=}, {b=} => c={pythagorean_theorem(a, b):.2f}".encode(FORMAT)
            )
        except ValueError:
            conn.send(f"'{msg}' is a wrong input.".encode(FORMAT))

        if msg == DISCONNECT_MESSAGE:
            connected = False

        print(f"[{addr}] {msg}")

    conn.close()

def start():
    print("Starting server...")
    server.bind(ADDR)
    server.listen()
    while True:
        conn, addr = server.accept()
```



```
        thread = threading.Thread(target=handle_client, args=(conn, addr))
        thread.start()

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

Приложение 2.2. Клиентская часть для задания 2.

```
import socket

from server import ADDR, FORMAT, DISCONNECT_MESSAGE, MAX_SIZE

def send(client: socket.socket, msg: str):
    message = msg.encode(FORMAT)
    client.send(message)

def start():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(ADDR)
    while True:
        msg = input("Input a, b separated by a comma (q - quit): ")
        if msg == "q":
            break
        send(client, msg)
        response = client.recv(MAX_SIZE).decode(FORMAT)
        print(f"Response from server: {response}")

    send(client, DISCONNECT_MESSAGE)
    print("Disconnected.")

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

Приложение 3.1. Серверная часть для задания 3.

```
import threading
import socket

ADDR = ("localhost", 5050)
FORMAT = "utf-8"
MAX_SIZE = 1024

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

def handle_client(conn: socket.socket, addr: tuple[str, int]):
    print(f"Connected: {addr}")

    with open("index.html", "r") as file:
        html_content = file.read()
    http_response = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n" + html_content
    conn.send(http_response.encode(FORMAT))
    conn.close()

def start():
    print("Starting server...")
    server.bind(ADDR)
    server.listen()
    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handle_client, args=(conn, addr))
        thread.start()

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

Приложение 3.2. Клиентская часть для задания 3.

```
import socket

from server import ADDR, FORMAT, MAX_SIZE

def start():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(ADDR)
    response = client.recv(MAX_SIZE).decode(FORMAT)
    print(response)
    print("Disconnected.")

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

Приложение 4.1. Серверная часть для задания 4.

```
import threading
import socket

ADDR = ("localhost", 5050)
FORMAT = "utf-8"
DISCONNECT_MESSAGE = "DISCONNECT"
MAX_SIZE = 1024

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
clients: set[socket.socket] = set()

def handle_client(conn: socket.socket, addr: tuple[str, int]):
    print(f"Connected: {addr}")

    connected = True
    while connected:
        msg = conn.recv(MAX_SIZE).decode(FORMAT)
        if not msg:
            break

        if msg == DISCONNECT_MESSAGE:
            connected = False

        for client in clients:
            client.sendall(f"From {addr[-1]}: {msg}".encode(FORMAT))

        print(f"[{addr}] {msg}")

    conn.close()

def start():
    print("Starting server...")
    server.bind(ADDR)
    server.listen()
    while True:
        conn, addr = server.accept()
        clients.add(conn)
        thread = threading.Thread(target=handle_client, args=(conn, addr))
        thread.start()

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

Приложение 4.2. Клиентская часть для задания 4.

```
import socket
```

```

from server import ADDR, FORMAT, DISCONNECT_MESSAGE, MAX_SIZE

def send(client: socket.socket, msg: str):
    message = msg.encode(FORMAT)
    client.send(message)

def start():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(ADDR)
    while True:
        msg = input("Input (q - quit): ")
        if msg == "q":
            break
        send(client, msg)

    send(client, DISCONNECT_MESSAGE)
    print("Disconnected.")

if __name__ == "__main__":
    start()

```

Приложение 4.3. Реализация многопользовательского чата для задания 4.

```

import socket

from server import ADDR, FORMAT

def connect():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(ADDR)
    return client

def start():
    connection = connect()
    while True:
        msg = connection.recv(1024).decode(FORMAT)
        print(msg)

if __name__ == "__main__":
    start()

```