МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчет по лабораторной №2

Почтовые протоколы

по дисциплине «Протоколы компьютерного взаимодействия»

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил | студент гр. ФИБ-4301-51-00 |  | Кочкин В.Р. |
| Проверил | преподаватель каф. ПМиИ |  | Белиц А.Б. |

Киров 2022

Цель работы

Познакомиться с почтовыми протоколами и реализовать получение и отправку сообщений.

Задания

**Задание 1**

Установите на виртуальную машину почтовый сервер, например, hMailServer (https://www.hmailserver.com/download). Настройте его для работы в режиме сети виртуального адаптера хоста. Дайте название домену (ветка Domains). Создайте почтовый аккаунт (ветка Accounts). Аутентификацию почтового сервера настройте без использования сертификатов.

**Задание 2**

Проверьте работоспособность почтового сервера используя клиент telnet (telnet или PuTTY).

**Задание 3**

В ходе лабораторной работы необходимо разработать программу на языке python, которая реализует функции отправки текстовых почтовых сообщений по протоколу SMTP.

**Задание 4**

В ходе лабораторной работы необходимо разработать программу на языке python которая реализует функции получения почтовых текстовых сообщений по протоколу POP3 или протоколу IMAP (на выбор).

**Задание 5**

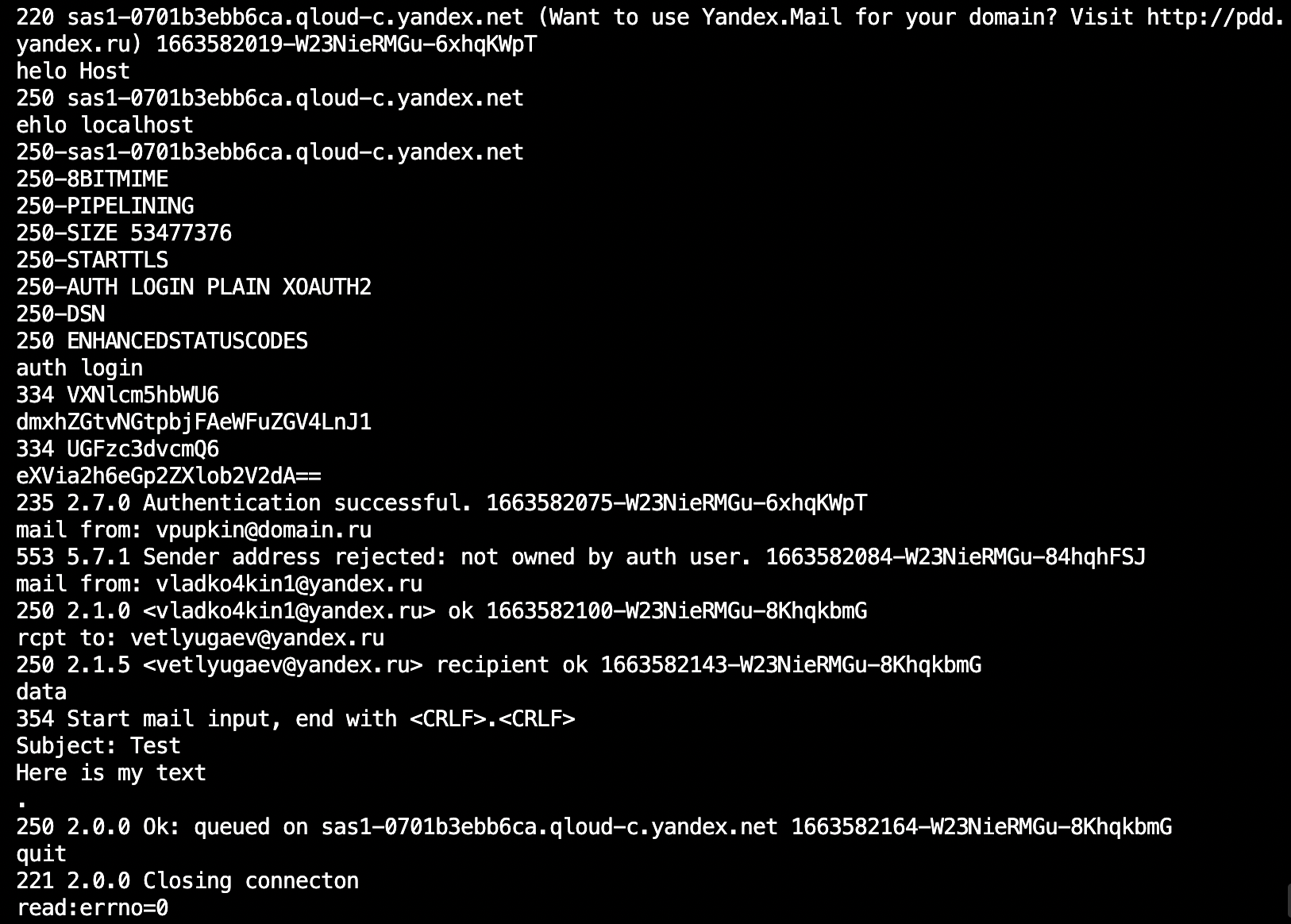
В ходе лабораторной работы необходимо разработать программу на языке python реализующую взаимодействие по протоколам POP3 и SMTP при помощи сокетов.

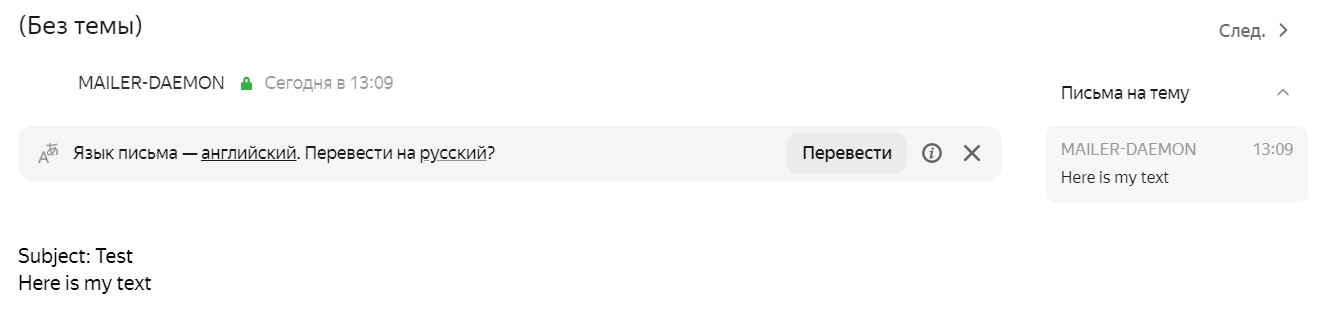
Полученные результаты

**Задание 1**

Сначала предполагалось использование почтового сервера Яндекса, но потом для упрощения работы был использован почтовый сервер одногруппника без сертификатов.

**Задание 2**

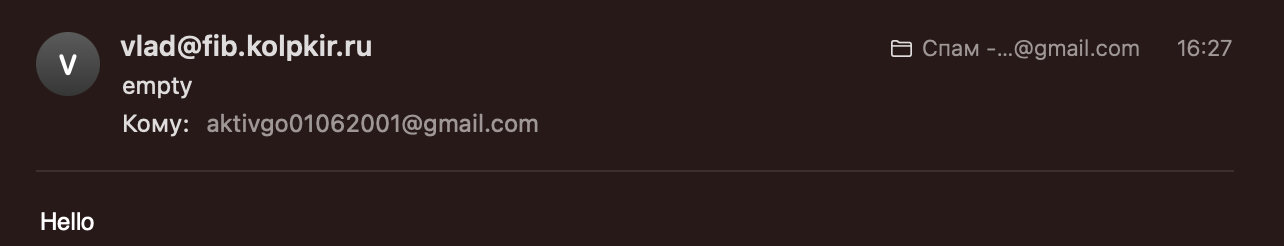




**Задание 3**

Изображение выглядит как текст

Автоматически созданное описание

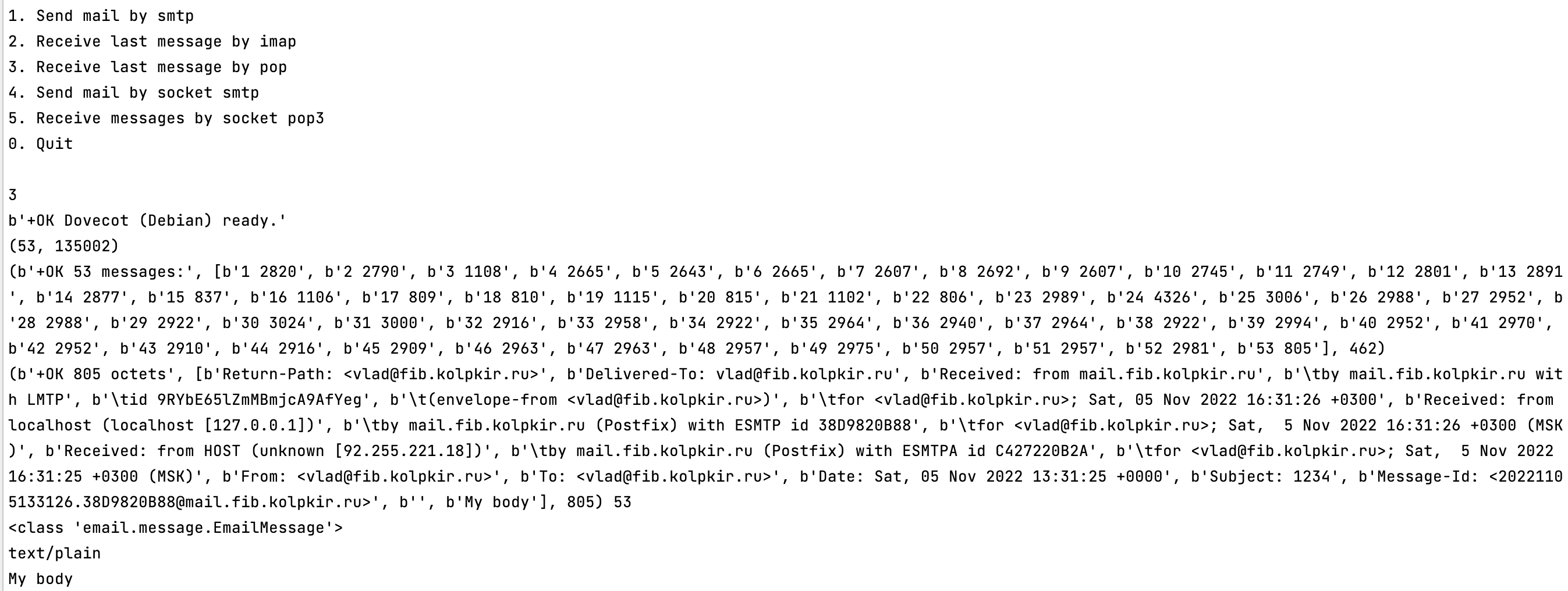


Листинг программы приведен в [приложении А Задание 3](#_Задание_1.).

**Задание 4**

Изображение выглядит как текст

Автоматически созданное описание



Листинг программы приведен в [приложении А Задание 4](#_Задание_4.).

**Задание 5**

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Листинг программы приведен в [приложении А Задание 5](#_Задание_5.).

Вывод

В ходе лабораторной работы я познакомился с почтовыми протоколами и реализовал получение и отправку сообщений.

# **Приложения**

## **Приложение А. Листинги программ**

Точка входа.

main.py

**import** os  
**import** time  
  
**from** smtp.client **import** SMTPClient  
**from** imap.client **import** IMAPClient  
**from** pop3.client **import** POP3Client  
**from** smtp.socket\_client **import** SocketSmtpClient  
**from** pop3.socket\_client **import** SocketPop3Client  
  
**from** email.mime.multipart **import** MIMEMultipart  
**from** email.mime.text **import** MIMEText  
  
HOST = os.getenv(**'HOST'**)  
  
  
**def** get\_auth():  
 **return** {  
 **'login'**: os.getenv(**'EMAIL\_LOGIN'**),  
 **'password'**: os.getenv(**'EMAIL\_PASSWORD'**)  
 }  
  
  
**def** create\_smtp\_message():  
 msg = MIMEMultipart()  
  
 body = input(**"input body: "**)  
  
 msg[**'From'**] = os.getenv(**"EMAIL\_LOGIN"**)  
 msg[**'Subject'**] = input(**"input subject: "**)  
  
 msg[**"Date"**] = time.strftime(**"%a, %d %b %Y %H:%M:%S %z"**)  
  
 msg[**'To'**] = input(**"input target: "**)  
  
 msg.attach(MIMEText(body, **'plain'**))  
  
 **return** msg  
  
  
**def** create\_socket\_smtp\_message():  
 msg = {}  
  
 body = input(**"input body: "**)  
 msg[**'body'**] = body  
  
 msg[**'from'**] = os.getenv(**"EMAIL\_LOGIN"**)  
 msg[**'subject'**] = input(**"input subject: "**)  
 msg[**'to'**] = input(**"input target: "**)  
  
 **return** msg  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 auth = get\_auth()  
 smtp = SMTPClient(HOST, auth)  
 imap = IMAPClient(HOST, auth)  
 pop3 = POP3Client(HOST, auth)  
 socket\_smtp = SocketSmtpClient(HOST, 25, auth)  
 socket\_pop3 = SocketPop3Client(HOST, 110, auth)  
  
 **while True**:  
 print(  
 **'1. Send mail by smtp\n'** +  
 **'2. Receive last message by imap\n'** +  
 **'3. Receive last message by pop\n'** +  
 **'4. Send mail by socket smtp\n'** +  
 **'5. Receive messages by socket pop3\n'** +  
 **'0. Quit\n'** )  
  
 choose = int(input())  
  
 **if** choose == 1:  
 message = create\_smtp\_message()  
 smtp.send\_mail(message)  
 print(**"successfully sent email to"**, message[**'To'**], **'\n'**)  
 **elif** choose == 2:  
 print(imap.receive\_last\_mail(), **'\n'**)  
 **elif** choose == 3:  
 print(pop3.receive\_last\_mail(), **'\n'**)  
 **elif** choose == 4:  
 message = create\_socket\_smtp\_message()  
 socket\_smtp.send\_mail(message)  
 print(**"successfully sent email to"**, message[**'to'**], **'\n'**)  
 **elif** choose == 5:  
 **while True**:  
 print(socket\_pop3.list())  
 choose = int(input())  
 **if** choose == 0:  
 **break** print(socket\_pop3.receive\_mail(choose))  
 **else**:  
 smtp.close()  
 pop3.close()  
 socket\_smtp.close()  
 socket\_pop3.close()  
 exit()

### Задание 3.

**import** smtplib  
  
  
**class** SMTPClient:  
 **def** \_\_init\_\_(self, host: str, auth: []):  
 self.smtp = smtplib.SMTP(host)  
 self.smtp.login(auth[**'login'**], auth[**'password'**])  
  
 **def** send\_mail(self, message):  
 self.smtp.sendmail(message[**'From'**], message[**'To'**], message.as\_string())  
  
 **def** close(self):  
 self.smtp.close()

### Задание 4.

IMAP:

**import** imaplib  
  
  
**class** IMAPClient:  
 **def** \_\_init\_\_(self, host: str, auth: []):  
 self.imap = imaplib.IMAP4(host)  
 self.imap.login(auth[**'login'**], auth[**'password'**])  
  
 **def** receive\_last\_mail(self):  
 self.imap.list()  
 self.imap.select(**"inbox"**)  
  
 result, data = self.imap.uid(**'search'**, **"ALL"**)  
  
 latest\_email\_uid = data[0].split()[-1]  
 result, data = self.imap.uid(**'fetch'**, latest\_email\_uid, **'(RFC822)'**)  
  
 **return** data[0][1]

POP3:

**import** poplib  
  
**from** email.parser **import** BytesParser  
**from** email.policy **import** default  
  
  
**class** POP3Client:  
 **def** \_\_init\_\_(self, host: str, auth: []):  
 self.pop3 = poplib.POP3(host)  
 self.pop3.user(auth[**'login'**])  
 self.pop3.pass\_(auth[**'password'**])  
  
 **def** receive\_last\_mail(self):  
 print(self.pop3.getwelcome())  
  
 stat = self.pop3.stat()  
 print(stat)  
  
 l = self.pop3.list()  
 print(l)  
  
 r = self.pop3.retr(len(l[1]))  
 print(r, len(l[1]))  
  
 bp = BytesParser(policy=default).parsebytes(b'**\r\n**'.join(r[1]))  
 print(type(bp))  
  
 **for** part **in** bp.walk():  
 print(part.get\_content\_type())  
 **if** part.get\_content\_maintype() == **'text'**:  
 **return** part.get\_content()  
  
 **def** close(self):  
 self.pop3.close()

### Задание 5.

POP3:

**from** socket **import** \*  
**import** base64  
**import** time  
  
  
**class** SocketPop3Client:  
 **def** \_\_init\_\_(self, host: str, port: int, auth: []):  
 self.client\_socket = socket(AF\_INET, SOCK\_STREAM)  
 self.\_\_initialize\_\_(host, port)  
 self.\_\_authorize\_\_(auth)  
  
 **def** \_\_initialize\_\_(self, host: str, port: int):  
 self.client\_socket.connect((host, port))  
 recv = self.client\_socket.recv(1024).decode()  
 **if** recv[:3] != **'+OK'**:  
 **raise '+OK reply not received from server.'  
  
 def** \_\_authorize\_\_(self, auth: []):  
 user\_msg = **'USER %s\r\n'** % auth[**'login'**]  
 self.client\_socket.send(user\_msg.encode())  
 recv = self.client\_socket.recv(1024).decode()  
 **if** recv[:3] != **'+OK'**:  
 **raise '+OK reply not received from server.'** pass\_msg = **'PASS %s\r\n'** % auth[**'password'**]  
 self.client\_socket.send(pass\_msg.encode())  
 recv = self.client\_socket.recv(1024).decode()  
 **if** recv[:3] != **'+OK'**:  
 **raise '+OK reply not received from server.'  
  
 def** list(self):  
 list\_msg = **'LIST\r\n'** self.client\_socket.send(list\_msg.encode())  
 recv = self.client\_socket.recv(1024).decode()  
 **if** recv[:3] != **'+OK'**:  
 **raise '+OK reply not received from server.'  
 return** recv  
  
 **def** receive\_mail(self, index: int):  
 retr\_msg = **'RETR %d\r\n'** % index  
 self.client\_socket.send(retr\_msg.encode())  
 recv = self.client\_socket.recv(1024).decode()  
 **if** recv[:3] != **'+OK'**:  
 **raise '+OK reply not received from server.'  
 return** recv  
  
 **def** close(self):  
 q = **"QUIT\r\n"** self.client\_socket.send(q.encode())  
 self.client\_socket.close()

SMTP:

**from** socket **import** \*  
**import** base64  
**import** time  
  
  
**class** SocketSmtpClient:  
 **def** \_\_init\_\_(self, host: str, port: int, auth: []):  
 self.client\_socket = socket(AF\_INET, SOCK\_STREAM)  
 self.\_\_initialize\_\_(host, port)  
 self.\_\_authorize\_\_(auth)  
  
 **def** \_\_initialize\_\_(self, host: str, port: int):  
 self.client\_socket.connect((host, port))  
 recv = self.client\_socket.recv(1024).decode()  
 **if** recv[:3] != **'220'**:  
 **raise '220 reply not received from server.'** helo\_command = **'EHLO HOST\r\n'** self.client\_socket.send(helo\_command.encode())  
 recv = self.client\_socket.recv(1024).decode()  
 **if** recv[:3] != **'250'**:  
 **raise '250 reply not received from server.'  
  
 def** \_\_authorize\_\_(self, auth: []):  
 base64\_str = (**'\x00'** + auth[**'login'**] + **"\x00"** + auth[**'password'**]).encode()  
 base64\_str = base64.b64encode(base64\_str)  
 auth\_msg = **'AUTH PLAIN '**.encode() + base64\_str + **'\r\n'**.encode()  
 self.client\_socket.send(auth\_msg)  
 recv = self.client\_socket.recv(1024).decode()  
 **if** recv[:3] != **'235'**:  
 **raise '235 reply not received from server.'  
  
 def** send\_mail(self, message):  
 mail\_from = **'MAIL FROM:<%s>\r\n'** % message[**'from'**]  
 self.client\_socket.send(mail\_from.encode())  
 recv = self.client\_socket.recv(1024).decode()  
 print(**'After MAIL FROM command: '** + recv)  
  
 rcpt\_to = **'RCPT TO:<%s>\r\n'** % message[**'to'**]  
 self.client\_socket.send(rcpt\_to.encode())  
 recv = self.client\_socket.recv(1024).decode()  
 print(**'After RCPT TO command: '** + recv)  
  
 data = **'DATA\r\n'** self.client\_socket.send(data.encode())  
 recv = self.client\_socket.recv(1024).decode()  
 print(**'After DATA command: '** + recv)  
  
 \_from = **'From: <%s>\r\n'** % message[**'from'**]  
 self.client\_socket.send(\_from.encode())  
  
 to = **'To: <%s>\r\n'** % message[**'to'**]  
 self.client\_socket.send(to.encode())  
  
 date = time.strftime(**'Date: %a, %d %b %Y %H:%M:%S +0000\r\n'**, time.gmtime())  
 self.client\_socket.send(date.encode())  
  
 subject = **"Subject: %s\r\n"** % message[**'subject'**]  
 self.client\_socket.send(subject.encode())  
  
 body = message[**'body'**] + **'\r\n.\r\n'** self.client\_socket.send(body.encode())  
  
 recv = self.client\_socket.recv(1024)  
 print(**'Response after sending message body: '** + recv.decode())  
  
 **def** close(self):  
 q = **"QUIT\r\n"** self.client\_socket.send(q.encode())  
 self.client\_socket.close()