

**SOCKET PROGRAMMING**

Lënda: Rrjetet Kompjuterike

Punoi: Arbër Hyseni ID:180714100058

Ideja e këtij projekti është programimi me soketa (*angl. Sockets*) ne gjuhën Python, të krijohet një kuptim I mirëfilltë se çfarë pune kryejnë soketat, përkatësisht çfarë roli kanë në rrjetet kompjuterike, si barten informatat ndërmjet serverit dhe klientit apo anasjelltas. Ne mund të punojmë me soketa duke e importuar modulin **socket**. Në pika të shkurtra soketi paraqet një pikë fundore (Angl. *Endpoint)*, e cila lejon komunikimin dykahor të dy programeve të cilat janë në të njejtin rrjet dhe me të është e “lidhur” numri specifik I portit, me anë të të cilit TCP layer e identifikon adresën e aplikacionit në të cilin është qëllimi që të dërgohen të dhëna.

Detyra që na është shtruar është dizajnimi I një arkitekture Klient-Server me protokollet TCP dhe UDP(*më shumë rreth tyre më vonë)*. Diçka e re që na është paraqitur është se si mundet Serveri të punoj me disa klienta njëkohësisht, pra të pranoj kërkesa dhe të dërgon kërkesa me shumë klienta njëkohësisht. E gjitha kjo është realizuar nëpërmjet parimit të punës së threads-ave (*angl. Thread)*.

Threads paraqesin një sekuencë të instruksioneve që mund të ekzekutohen pavarësisht kodit tjetër ne atë program, apo sikur literaturës kryesore angleze që I përmbahemi në ligjerata, threadi mund ta paramendojmë si një sub-proces I lehtë I programit. Një multi-thread program d.m.th ka dy ose më shumë pjesë të cilat mund të punojnë njëkohësisht. Sa herë që një klient lidhet me Serverin duhet të krijohet një thread I ri për Server. Për të punuar me Threads së pari duhet importuar modulin **threading** dhe pastaj klasën [**threading.Thread**](http://faculty.salina.k-state.edu/tim/NPstudy_guide/servers/threads.html#threading.Thread). Në qoftë se dojmë ta ndajmë serverin në pjesë për ta kuptuar më mirë, ndarja do të ishte kështu:

* Ndëgjo dhe prano lidhje me soketa
* Krijo dhe starto threads-at “fëmijë”
* Krijo një infinit loop

Serveri me protokollin TCP dhe serveri me protokollin UDP duhet që të ketë metodat e definuara për zgjidhjen e problemeve të dhëna, dhe të pranoj një input nga Klienti, të përpunoj të dhënat dhe të dërgoj mesazhin e duhur te Klienti. Si p.sh. kështu më poshtë:

Operacioni (IPADDRESS, PORT, COUNT, REVERSE, PALINDROME, TIME, GAME, GCF, CONVERT, PASSWORDSTRENGTH, IPCLASS)? TIMEPërgjigjja: 10.04.2020 11:00:00

Që të kuptojmë se si kjo arkitekturë e serverit funksionon më shumë duhet kuptuar konceptet e TCP-së dhe UDP-së, si dhe te soketave.

* **Transmission Control Protocol** apo shkurt TCP është një nga protokollet kryesore të suitës së protokolleve të Internetit. Ky protokoll buron së pari nga zbatimet fillestare të rrjetit në të cilin zhvilloi dukshëm Protokollin e Internetit (IP). Prandaj, e gjithë suita zakonisht quhet TCP/IP. TCP siguron shpërndarje të besueshme, të renditur dhe të kontrolluar nga gabimet e një rrjedhe (*Angl. Stream)* oktetesh (bajtësh) midis aplikacioneve që funksionojnë në hostet që komunikojnë përmes një rrjeti IP. Aplikacione të mëdha të internetit të tilla si World Wide Web, email, administrimi i largët dhe transferimi i file-ve mbështeten në TCP, e cila është pjesë e shtresës së transportit(Angl. Transport Layer) të kompletit TCP/IP. TCP është I orientuar në konektivitet, një lidhje në mes të serverit dhe klientit duhet realizuar para së të dërgohen të dhënat.
* **User Datagram Protocol** (UDP) është një nga anëtarët kryesorë të grupit të protokollit të Internetit. Aplikacionet kompjuterike mund të dërgojnë mesazhe, në këtë rast të referuara si datagrame, te hostet e tjera në një rrjet të Protokollit të Internetit (IP). Komunikimet paraprake nuk kërkohen për të vendosur kanale komunikimi ose shtigje të të dhënave. UDP përdoret më së shumti për të krijuar lidhje të cilat karakterizohen me “low latency” dhe “loss tolerating” ndërmjet aplikacioneve në internet. UDP konsiderohet protokoll “connectionless” për arsye se nuk ka nevojë të krijoj një qark virtual të realizohet para se të dergohen të dhëna.

Dallimet fundamentale ndërmjet TCP dhe UDP janë të paraqitura në tabelën e mëposhtme:

|  |  |  |
| --- | --- | --- |
| Karakteristikat | UDP | TCP |
| Analogjia | Dergimi me postë | Telefonatë |
| Faza e komunikimit | Nuk ekziston | Ekziston |
| Informatat | Individuale | Sekuenciale |
| Informata të humbura | Klienti duhet t’I pret humbjet | Protokolli nuk lejon *(Angl. Reliable)* |
| Radhitja e informatave | Nuk mbahet radhë | Mbahet radhë |
| Faza e shkyçjës | Nuk ekziston | Ekziston |
| Kompleksiteti | I ultë | I lartë |

*Burimi: University of Glamorgan dhe Wikipedia*

**Problemet dhe sfidat**

Sa I përket këtij projekti sado që mund të mendojë dokush që gjuha Python do ishte problemi kryesor, nuk është e vërtetë pasi që gjuha Python është një gjuhë shumë e lehtë për tu kuptuar sidomos pas mësimit të gjuhëve të tjera si Java dhe C++, por në fakt për mua ka qenë një sfidë e vogël implementimi I Threads-ave në arkitekturën e serverit TCP dhe UDP, megjithatë me ndihmën e resurseve të mjaftueshme në Internet, edhe kjo sfidë u tejkalua.

Mandej është dashur që të implementohen një varg I tërë I funksioneve dhe importimit të moduleve të ndryshme që Python I ka të gatshme për ne, për arsye të ndryshme, të kalkulimit, manipulimit, reprezentimit të të dhënave, hyrjes në input etj. Në lidhje me threads, kam krijuar një klasë si kjo

class ClientThread(threading.Thread):

    def \_\_init\_\_(self,clientAddress,clientsocket):

        threading.Thread.\_\_init\_\_(self)

        self.csocket = clientsocket

        print ("New connection added: ", clientAddress)

def run(self):

        self.csocket.send(bytes("Hi...",'utf-8'))

kjo klasë nuk bën asgjë më shumë se implementimin e Threads dhe përdorimin e metodave specifike sikrijimin e metodës **\_\_init\_\_** që t’a bëjmë “override” metodën me argumentë shtesë sipas nevojës, në këtë rast me **clientAddress dhe clientsocket**, pastaj të bëjmë “override” metodën run që të implementojmë se çfarë të bëjë thread-I kur të startohet (*më shumë për* ***run(self [,args])*** *pak më vonë*). Për çdo klient që lidhet me Server, kriohet një thread I ri, pasi që të krijojmë klasën Thread, mund ta startojmë atë me metoden **start(),** dhe me këtë përfundon konfigurimi I threads-ave. Konfugirimi I kominikimit Klient-Server është pothuajshe I ngjashëm me atë që e kemi bërë në klasë, bashkë me threads këto mundësojnë, një server funksional dhe pa ndonjë gabim në ekzekutim të kodit. Në mënyrën se si e kam përshtatur unë komunikimin Klient-Server, funksionon ashtu që serveri pas lidhjes me klientin formon një sesion komunikimi gjatë së cilës klienti mund të dërgoj një komandë dhe ajo përpunohet nga serveri pastaj dërgohet përsëri te klienti me përgjigjjen e duhur, pas kësaj kemi opsionin të vahzdojmë më tutje apo jo, nëse jo atëherë ndërprehet lidhja me serverin.

Te UDP arkitektura, është kërkuar që pas një komande të Klientit të ndërpritet komunikimi.

Që të kemi një pamje se si realizohet një konektim I Serverit me Klientin na nevojiten këta rreshta kod:

**SERVERI:**

class ClientThread(threading.Thread): # Thread Class

   def \_\_init\_\_(self,clientAddress,clientsocket): # Konstruktori init me te dhenat, ku self paraqet instancat e vet klases

      threading.Thread.\_\_init\_\_(self)

      self.csocket = clientsocket

      print ("New connection added: ", clientAddress)

   def run(self): # Funksioni run qe ekzekuton Clienthread sa here qe krijohet nje

      self.csocket.send(bytes("Hi...",'utf-8'))

      msg = ''

      while True:

         data = self.csocket.recv(2048) # Receive data from the socket(bytes)

         msg = data.decode() #from bytes to string

         print ("Input from client: ", msg)

         self.csocket.send(bytes(msg,'UTF-8')) # Send data to the socket

LOCALHOST = "localhost"

PORT = 13000

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) # create socket object

server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1) # get and set  options on sockets

server.bind((LOCALHOST, PORT)) # bind socket to address

print("Server started")

print("Waiting for client request..")

while True:

    server.listen(1) # 1 queue at a time

    clientsock, clientAddress = server.accept() # Accept the connection

    newthread = ClientThread(clientAddress, clientsock) #Informata per threads

    newthread.start() # Starto threadin e krijuar

Ku pothuajse secili rresht është përshkuar në kod.

Që të realizohet lidhja e Klientit me Server na duhet edhe kodi I pjesërishëm I FIEKTCP-Klient.py

**Klienti:**

import socket

SERVER = "localhost"

PORT = 13000

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) # create socket object

client.connect((SERVER, PORT)) # Connect to a remote socket at given address

while True:

  serverdata =  client.recv(1024) # Receive data from the socket(bytes)

  print("From Server :" ,serverdata.decode())

  userinput = input()

  client.sendall(bytes(userinput,'UTF-8')) # Send data to the socket, send until no more data left

Ky nuk është kodi I tërë, në faqet në vijiim e shohim edhe kodin e tërë.

Nuk përjashtohet mundësia që çdo funksion/metodë do të mund të formohej në mënyra dhe forma të ndryshme, por unë kam zgjedhur një menyrë që më ka pelqyer personalisht.

Për shembull për krijimin e threads do të ishte e mundur që për çdo çasje të klientit të krijohej manualisht një **new Thread(),** apo metoda *reverse* pa mënyren e *Slicing* do të mund të kryhej me një for loop sikur kjo:

s1 = ''

    for c in s:

        s1 = c + s1 #append(shto)

E shumë e shumë menyra tjera për secilin funksion në listën e funksioneve.

Sa I përket funksioneve në faqen tjetër do të shpjegohen të gjitha.

**Funksionet:**

* Para se të vazhdojmë me shpjegimin e funksioneve, vlen të ceket edhe një herë se çdo funksion I mëposhtëm mund të zhvillohet në mënyra të ndryshme, varësisht se cila I duket studentit më e përshtatshme, disa funksione specifike sikur ajo re.sub, [::-1] apo slicing, I kam hulumtuar dhe I kam gjetur në faqe të ndryshme të internetit që do mundohem të I inkuadroj në sektorin e bibliografisë në fund të raportit.

**Funksioni regex()**

import re

#client input validation

def regex(string):

    string = string.lower()

    string = re.sub(' +', ' ', string)

    string = string.lstrip() #Left strip to remove whitespaces

    return string.split(' ') #Split for splitting input in spaces

Për përpunimin e inputit te klientit shërben funksioni i mësipërm, I cili sipas meje është një funksion shumë kreativ dhe I qëlluar për arsyjen që përdoret. Përpos komenteve në gjuhën angleze ky funksion ka një parametër hyrës d.m.th (inputin e klientit), I cili pastaj përpunohet më tutje me funksionet përkatese siç është ajo sub dhe split, të cilat I përkasin modulit re, metoda **sub()** kthen një string në baze të kushtit më të majtë, pastaj bëhet **lstrip()** në rast se stringu I kthyer ka një whitespace në anën e majtë të stringut përkatës, duke vazhduar me funksionin **split()** e cila krijon një listë në bazë të kushtit brenda kllapave.

**Funksioni COUNT()**

def COUNT(value):

    if(value.isalpha()): #A eshte shkronje alfabetike

        vow = 0

        con = 0

        list1 = ["a", "e", "i", "o", "u"] #krijimi i nje liste

        for char in value:

            if char in list1: #nese char gjendet ne liste

                vow+=1

            elif char not in list1 and char>="a" and char<="z":

                con+=1

    else:

        raise Exception("Lejohen vetem shkronja te alfabetit.")

    result = f'Consonant {con} Vowel {vow}' #return permes f string

    return result

Siç po shihet metoda e mësipërme, shërben për njehsimin e zanoreve dhe bashkëtingëlloreve të stringut që është në hyrje të funksionit. Brenda if statement kam përdorur një funksion **isalpha()** I cili vërteton se a është çdo karakter I stringut shkronjë alfabetike, dhe nëse jo atëherë ndodh një **Exception** me mesazhin e caktuar dhe rezultati përmes metodës së **f-stringjeve** e cila është një risi ne Python 3, kam krijuar një “rezultat” I cili pasohet mandej në program main dhe te klienti.

**Funksioni REVERSE()**

def REVERSE(s):

    if(s.isalpha()):

        revres = s[::-1]

    elif not (s.isalpha()):

        #Show exception on terminal for invalid input

        raise Exception("Only words from the alphabet are allowed")

Ky funksion është një funksion I shkurtër dhe poashtu shumë efikas, pasi që edhe ka error handling por edhe përdor një metodë e cila quhet **slicing** dhe rasti konkret ne rreshtin e tretë të kodit të mësipërm, përdoret slicing për të kthyer stringun hyrës të funksionit mbrapsht.

**Funksioni PALINDROME()**

def Palindrome(s):

    true = "True"

    false = "False"

    rev = s[::-1] #slicing method to reverse given string

    if(s.isalpha()):

        if (s == rev):

            return true

        else:

            return false

    else:

        result = "Only numbers allowed"

        return result

Funksioni palindrome ka për detyrë që kur klienti të jep komandën palindrome *stringicfaredoshem,* të marr një përgjigjje se a është stringu që e ka dhënë një palindrom, d.m.th.: se a është I njejti edhe kur kthehet mbrapsht, sikur psh fjala *kimik*. Kjo është arritur duke përdorur përsëri slicing, dhe është trajtuar paraqitja eventuale e gabimeve, do të ishte mirë të implementonim raise Exception etj., mirëpo pasi që arkitektura e këtij programi është delikat ajo ndërpren workflowin e duhur të programit.

**Funksioni GAME()**

from random import seed, randint

def GAME():

    lst = []

    forseed = randint(1, 100) # random number generator

    seed(forseed)

    for \_ in range(5):

        rvalue = randint(1, 35) # generate random numbers between 1 and 35

        if rvalue not in lst: # mos te kemi duplikate

lst.append(rvalue) # append to list

    return lst

Funksioni GAME ka për qëllim që të krijoj një listë me 5 numra arbitrar, me kufi prej 1 deri në 35. Kjo është realizuar me përdorimin e modulit random, ku prej saj janë përdorur funksionet built-in **seed()** dhe **randint().** Që ky funksion të krijoj njëmend numra të çfarëdoshëm duhet të perdoret funksioni seed() I cili gjeneron një numër(parametër) që inicializon një sekuencë të numrave dhe së bashku me përdorimin e randint() do të krijohet një varg me numra arbitrar që nuk përsëriten.

**Funksioni TIME()**

from datetime import datetime

def Time(): #ketu hyhet ne perdorim librari e gatshme datetime

    time = datetime.now()

    date\_string = time.strftime("%d/%m/%Y %H:%M:%S") #string format for time

    return date\_string

Vetë emri I funksionit tregon se çfarë bën, pra kthen orën e saktë por edhe datën, në të cilën kërkohet ky funksion nga klienti. Kjo është bërë e mundur nga moduli datetime, ku ky modul na jep klasa te gatshme për manipulimin e datave dhe kohës. Për këtë funksion ndihmë ka qenë faqja docs.python. Për format të stringut është përdorur metoda strftime() që kthen datën dhe kohën e formatuar në bazë të kushteve brenda kllapave të funksionit.

**Funksioni GCF**

def GCF(x, y):

    if x > y:

        little = y

    else:

        little = x

    for i in range(1, little+1): #for loop prej 1 deri te me i vogli +1

        if ((x % i == 0) and (y % i == 0)): #operacioni modulo

            gcf = i

    return gcf

Funksioni GCF do të thotë funksioni **greatest common factor** I cili ka për detyrë të kthejë faktorin më të madh të përbashkët të dy numrave që janë në hyrje të funksionit, kjo është bërë e mundur përmes këtij algoritmi relativisht të thjeshtë ku me një for-loop prej 1 deri te numri më vogel në hyrje të funksionit, shiqojmë me ndihmën e operacionit modulo dhe me anë të njohurive elementare të matematikës të gjejmë faktorin më të madh të përbashkët të ata dy numrave në hyrje.

Dy funksionet *extra* që I kam bërë janë**:**

* **PasswordStrength**
* **Fibonacci**

**Funksioni PasswordStrenth**

import re

def passwordStrength(s):

    # Enter password text

    length = re.compile(r'(\w{8,})')  # Check if password has atleast 8 characters

    lower = re.compile(r'[a-z]+') # Check if at least one lowercase letter

    digit = re.compile(r'[0-9]+') # Check if at least one digit.

    special = re.compile(r'[!@#$%^&\*\_.]+')

    if length.findall(s) == []:  # Checks if the password length is not 8 characters long, findall() kthen string array

        result = 'Your Password must contain at least 8 characters'

    elif lower.findall(s)==[]: # Checks if the password does not contain a  lowercase character

        result = 'Your Password must contain at least one lowercase character'

    elif digit.findall(s)==[]: # Checks if the password does not contain a  digit character

        result = 'Your Password must contain at least one digit character'

    elif special.findall(s)==[]: # Checks if the password does not contain a  special char

        result = 'Your Password must contain at least one special character'

    else:  # if the above 4 conditions are successfully passed, pringt out a message saying the password is strong.

        result = 'Your password is strong, congratulations!'

    return result

Si funksion extra që është dashur t’a bëjmë, unë kam përzgjedhur funksionin të ashtuquajtur **passwordStrength,** nga vetë emri tregohet se teston nje string hyrës se a është password I sigurt apo jo, dhe nëpërmjet built-in funksioneve te modulit re(*regular expression*) siç është re.compile() e cila kompajllon një “mostër” të regular expression në regular expression objekt, dhe për të punuar me atë objekt ne manipulojmë me funksionet **.findall()** apo **.match()**. Në këtë rast me .findall() kemi kërkuar se a ka mjaftueshëm karaktere, numra, etj. Pastaj me if, elif, else statements kemi manipuluar me të dhënat që I kemi.

**Funksioni Fibonacci :**

def Fibonacci(n):

   if n<0:

      print("No such number allowed!")

   # First Fibonacci number is 0 (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...)

   elif n==1:

      return 0

   # Second Fibonacci number

   elif n==2:

      return 1

   else:

      return Fibonacci(n-1)+Fibonacci(n-2)

***Fibonacci numbers***është një term mjaft I njohur në botën e programimit dhe matematikës, dhe paraqet një sekuencë të numrave e cila krijohet nga shuma e numrit të parë I cili është 0 dhe 1 pastaj shkon duke u rritur, 0 + 1 = 1, pra 0 1 1, pastaj 1 + 1 = 2, e kështu me radhë. Në kodin e mësipërm mund të themi se është paraqitur diçka më ndryshe se funksionet e tjera, bëhet fjalë për **rekurzionin** dhe me anë të rekurzionit është krijuar një algoritëm shumë I mirë për gjetjen e n-të të numrit në sekuencën e *Fibonaccit*. Pra nëse jepim numrin 10, na kthen 34.

**Kodi, rezultatet dhe figurat**

**Më poshtë do të listoj kodet për që të 4tat file, FIEK TCP Server, FIEK TCP Client, FIEK UDP Server, FIEK UDP Client**

* **FIEK TCP Server.py**
* import socket, threading, re
* from random import seed, randint
* from datetime import datetime
* #client input validation
* def regex(string):
* string = string.lower()
* string = re.sub(' +', ' ', string)
* string = string.lstrip() #Left strip to remove whitespaces
* return string.split(' ') #Split for splitting input in spaces
* def COUNT(value):
* if(value.isalpha()): #A eshte shkronje alfabetike
* vow = 0
* con = 0
* list1 = ["a", "e", "i", "o", "u"] #krijimi i nje liste
* for char in value:
* if char in list1: #nese char gjendet ne liste
* vow+=1
* elif char not in list1 and char>="a" and char<="z":
* con+=1
* else:
* return "Lejohen vetem shkronja te alfabetit."
* result = f'Consonant {con} Vowel {vow}' #return permes f string
* return result
* def cmToFeet(value):
* ft = float(value)\*0.032802
* result = "Feet= %.2f" % ft
* return result
* def feetToCm(value):
* cm = float(value/0.032802)
* result = "Cm= %.2f" % cm
* return result
* def kmToMiles(value):
* miles = value \* 0.621371
* result = "Miles= %.2f" % miles
* return result
* def milesToKm(value):
* km = value/0.621371
* result = "Km= %.2f" % km
* return result
* def REVERSE(s):
* if(s.isalpha()):
* revres = s[::-1]
* elif not (s.isalpha()):
* #Show exception on terminal for invalid input
* return "Only words from the alphabet are allowed"
* return revres
* def Palindrome(s):
* true = "True"
* false = "False"
* rev = s[::-1] #slicing method to reverse given string
* if(s.isalpha()):
* if (s == rev):
* return true
* else:
* return false
* else:
* result = "Only numbers allowed"
* return result
* def GAME():
* lst = []
* forseed = randint(1, 100) #random number generator
* seed(forseed)
* for \_ in range(5):
* rvalue = randint(1, 35) #generate random numbers between 1 and 35
* lst.append(rvalue) #append to list
* return lst
* def Time(): #ketu hyhet ne perdorim librari e gatshme datetime
* time = datetime.now()
* date\_string = time.strftime("%d/%m/%Y %H:%M:%S") #string format for time
* return date\_string
* #Find greatest common factor
* def GCF(x, y):
* if x > y:
* little = y
* else:
* little = x
* for i in range(1, little+1): #for loop prej 1 deri te me i vogli +1
* if ((x % i == 0) and (y % i == 0)): #operacioni modulo
* gcf = i
* return gcf
* def passwordStrength(s):
* # Enter password text
* length = re.compile(r'(\w{8,})')  # Check if password has atleast 8 characters
* lower = re.compile(r'[a-z]+') # Check if at least one lowercase letter
* digit = re.compile(r'[0-9]+') # Check if at least one digit.
* special = re.compile(r'[!@#$%^&\*\_.]+')
* if length.findall(s) == []:  # Checks if the password does not contain 8 characters, findall() kthen string array
* result = 'Your Password must contain at least 8 characters'
* elif lower.findall(s)==[]: # Checks if the password does not contain a lowercase character
* result = 'Your Password must contain at least one lowercase character'
* elif digit.findall(s)==[]: # Checks if the password does not contain a digit character
* result = 'Your Password must contain at least one digit character'
* elif special.findall(s)==[]: # Checks if the password does not contain a special char
* result = 'Your Password must contain at least one special character'
* else:  # if the above 4 conditions are successfully passed, pringt out a message saying the password is strong.
* result = 'Your password is strong, congratulations!'
* return result
* def Fibonacci(n):
* if n<0:
* print("No such number allowed!")
* # First Fibonacci number is 0 (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...)
* elif n==1:
* return 0
* # Second Fibonacci number
* elif n==2:
* return 1
* else:
* return Fibonacci(n-1)+Fibonacci(n-2)
* class ClientThread(threading.Thread):
* def \_\_init\_\_(self,clientAddress,clientsocket):
* threading.Thread.\_\_init\_\_(self)
* self.csocket = clientsocket
* print ("New connection added: ", clientAddress)
* def run(self): #qekjo run(self) edhe init ma nalt spo e kom bash te qarte qa bojn, e po ma hupin zumin qito hahahaha
* self.csocket.send(bytes("Hi...",'utf-8'))
* msg = ''
* flist = ["count", "convert", "game", "palindrome", "gcf", "reverse", "time", "hi", "Hi", "fibonacci", "passwordstrength"]
* while True:
* data = self.csocket.recv(2048) #ose 1024
* msg = data.decode()
* vinput = regex(msg)
* if vinput[0] == 'ipaddress':
* exe = str(clientAddress[0])
* self.csocket.send(bytes(exe,'UTF-8'))
* print ("Received from client: ", msg)
* elif vinput[0] == 'port':
* exe = str(clientAddress[1])
* self.csocket.send(bytes(exe,'UTF-8'))
* print ("Received from client: ", msg)
* elif(vinput[0] == 'count'):
* exe = COUNT(vinput[1])
* self.csocket.send(bytes(exe,'UTF-8'))
* print ("Received from client: ", msg)
* elif(vinput[0] == 'reverse'):
* print ("Received from client: ", msg)
* exe = REVERSE(vinput[1])
* self.csocket.send(bytes(exe,'UTF-8'))
* elif(vinput[0] == 'palindrome'):
* print ("Received from client: ", msg)
* exe = Palindrome(vinput[1])
* self.csocket.send(bytes(exe,'UTF-8'))
* elif(vinput[0] == 'time'):
* print ("Received from client: ", msg)
* exe = Time()
* self.csocket.send(bytes(exe,'UTF-8'))
* elif(vinput[0] == 'game'):
* print ("Received from client: ", msg)
* exe = GAME()
* listToStr = ' '.join(map(str, exe)) #list to string
* self.csocket.send(bytes(listToStr,'UTF-8'))
* elif(vinput[0] == 'gcf'):
* if(vinput[1].isdigit() and vinput[2].isdigit()):
* print ("Received from client: ", msg)
* exe = GCF(int(vinput[1]), int(vinput[2]))
* exes = str(exe) #int to str
* self.csocket.send(bytes(exes,'UTF-8'))
* else:
* print ("Received from client: ", msg)
* self.csocket.send(bytes("Only Integers allowed",'UTF-8'))
* elif(vinput[0] == 'convert' and vinput[1] == 'cmtofeet'): #looks ugly but no other option!
* print ("Received from client: ", msg)
* if vinput[2].isdigit():
* exe = cmToFeet(float(vinput[2]))
* self.csocket.send(bytes(exe,'UTF-8'))
* else:
* self.csocket.send(bytes("Only numbers allowed",'UTF-8'))
* elif(vinput[0] == 'convert' and vinput[1] == 'feettocm'):
* print ("Received from client: ", msg)
* if vinput[2].isdigit():
* exe = feetToCm(float(vinput[2]))
* self.csocket.send(bytes(exe,'UTF-8'))
* else:
* self.csocket.send(bytes("Only numbers allowed",'UTF-8'))
* elif(vinput[0] == 'convert' and vinput[1] == 'kmtomiles'):
* print ("Received from client: ", msg)
* if vinput[2].isdigit():
* exe = kmToMiles(float(vinput[2]))
* self.csocket.send(bytes(exe,'UTF-8'))
* else:
* self.csocket.send(bytes("Only numbers allowed",'UTF-8'))
* elif(vinput[0] == 'convert' and vinput[1] == 'milestokm'):
* print ("Received from client: ", msg)
* if vinput[2].isdigit():
* exe = milesToKm(float(vinput[2]))
* self.csocket.send(bytes(exe,'UTF-8'))
* else:
* self.csocket.send(bytes("Only numbers allowed",'UTF-8'))
* elif(vinput[0] == 'passwordstrength'):#extra
* asteriks = '\*' \* len(vinput[1])
* print ("Received from client: " + vinput[0] + ' ' + asteriks)
* exe = passwordStrength(vinput[1])
* self.csocket.send(bytes(exe,'UTF-8'))
* elif(vinput[0] == 'fibonacci'):
* print("Recieved from client: ", msg)
* if (vinput[1].isdigit()):
* exe = str(Fibonacci(int(vinput[1])))
* self.csocket.send(bytes(exe,'UTF-8'))
* else:
* self.csocket.send(bytes("Only integers allowed!",'UTF-8'))
* elif vinput[0]=='bye':
* print ("From client", msg)
* self.csocket.send(bytes(msg,'UTF-8'))
* break
* elif vinput[0] not in flist:
* print(vinput[0])
* exe = "<error>"
* self.csocket.send(bytes(exe,'UTF-8'))
* print ("Client at ", clientAddress , " disconnected...")
* LOCALHOST = "localhost"
* PORT = 13000
* #The socket() call creates a socket in the specified domain and of the specified type.
* server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) #AF\_INET = Internet domain, SOCK\_STREAM = stream socket for TCP protocol
* server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1) #The setsockopt function sets a socket option.
* server.bind((LOCALHOST, PORT)) #bind() associates the socket with its local address
* print("Server started")
* print("Waiting for client request..")
* while True:
* server.listen(1)
* clientsock, clientAddress = server.accept()
* newthread = ClientThread(clientAddress, clientsock)
* newthread.start()
* **FIEK TCP Client.py**
* import socket
* SERVER = "localhost"
* PORT = 13000
* client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  # create socket object
* client.connect((SERVER, PORT))# Connect to a remote socket at given address
* client.sendall(bytes("Hi",'UTF-8')) # send Hi message to socket
* while True:
* serverdata = client.recv(1024) #receive welcome message
* print("From Server:" ,serverdata.decode())
* clientdata = input("Operacioni (IPADDRESS, PORT, COUNT, REVERSE, PALINDROME, TIME, GAME, GCF, CONVERT, PASSWORDSTRENGTH, FIBONACCI): ")
* client.sendall(bytes(clientdata,'UTF-8'))
* serverdata = client.recv(2048) # get the server data
* print("Pergjigjja: " ,serverdata.decode())
* while True:
* answer = input('Run again? (p/j): ')
* if answer == 'p':
* clientdata = input("Operacioni (IPADDRESS, PORT, COUNT, REVERSE, PALINDROME, TIME, GAME, GCF, CONVERT, PASSWORDSTRENGTH, FIBONACCI): ")
* client.sendall(bytes(clientdata,'UTF-8'))
* serverdata = client.recv(1024)
* print("Pergjigjja: " ,serverdata.decode())
* continue
* elif answer == 'j':
* print ('Goodbye')
* client.sendall(bytes("bye",'UTF-8'))
* break
* break
* client.close() # close connection
* print("Connection with server is done")
* **FIEK UDP Server.py**
* import threading, socket, re
* from random import seed, randint
* from datetime import datetime
* def regex(string):
* string = string.lower()
* string = re.sub(' +', ' ', string)
* string = string.lstrip() #Left strip to remove whitespaces
* return string.split(' ') #Split for splitting input in spaces
* def COUNT(value):
* if(value.isalpha()): #A eshte shkronje alfabetike
* vow = 0
* con = 0
* list1 = ["a", "e", "i", "o", "u"] #krijimi i nje liste
* for char in value:
* if char in list1: #nese char gjendet ne liste
* vow+=1
* elif char not in list1 and char>="a" and char<="z":
* con+=1
* else:
* return "Lejohen vetem shkronja te alfabetit."
* result = f'Consonant {con} Vowel {vow}' #return permes f string
* return result
* def cmToFeet(value):
* ft = float(value)\*0.032802
* result = "Feet= %.2f" % ft
* return result
* def feetToCm(value):
* cm = float(value/0.032802)
* result = "Cm= %.2f" % cm
* return result
* def kmToMiles(value):
* miles = value \* 0.621371
* result = "Miles= %.2f" % miles
* return result
* def milesToKm(value):
* km = value/0.621371
* result = "Km= %.2f" % km
* return result
* def REVERSE(s):
* if(s.isalpha()):
* revres = s[::-1]
* elif not (s.isalpha()):
* #Show exception on terminal for invalid input
* return "Only words from the alphabet are allowed"
* return revres
* def Palindrome(s):
* true = "True"
* false = "False"
* rev = s[::-1] #slicing method to reverse given string
* if(s.isalpha()):
* if (s == rev):
* return true
* else:
* return false
* else:
* result = "Only numbers allowed"
* return result
* def GAME():
* lst = []
* forseed = randint(1, 100) #random number generator
* seed(forseed)
* for \_ in range(5):
* rvalue = randint(1, 35) #generate random numbers between 1 and 35
* lst.append(rvalue) #append to list
* return lst
* def Time(): #ketu hyhet ne perdorim librari e gatshme datetime
* time = datetime.now()
* date\_string = time.strftime("%d/%m/%Y %H:%M:%S") #string format for time
* return date\_string
* #Find greatest common factor
* def GCF(x, y):
* if x > y:
* little = y
* else:
* little = x
* for i in range(1, little+1): #for loop prej 1 deri te me i vogli +1
* if ((x % i == 0) and (y % i == 0)): #operacioni modulo
* gcf = i
* return gcf
* def passwordStrength(s):
* # Enter password text
* length = re.compile(r'(\w{8,})')  # Check if password has atleast 8 characters
* lower = re.compile(r'[a-z]+') # Check if at least one lowercase letter
* digit = re.compile(r'[0-9]+') # Check if at least one digit.
* special = re.compile(r'[!@#$%^&\*\_.]+')
* if length.findall(s) == []:  # Checks if the password does not contain 8 characters, findall() kthen string array
* result = 'Your Password must contain at least 8 characters'
* elif lower.findall(s)==[]: # Checks if the password does not contain a lowercase character
* result = 'Your Password must contain at least one lowercase character'
* elif digit.findall(s)==[]: # Checks if the password does not contain a digit character
* result = 'Your Password must contain at least one digit character'
* elif special.findall(s)==[]: # Checks if the password does not contain a special char
* result = 'Your Password must contain at least one special character'
* else:  # if the above 4 conditions are successfully passed, pringt out a message saying the password is strong.
* result = 'Your password is strong, congratulations!'
* return result
* def Fibonacci(n):
* if n<0:
* print("No such number allowed!")
* # First Fibonacci number is 0 (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...)
* elif n==1:
* return 0
* # Second Fibonacci number
* elif n==2:
* return 1
* else:
* return Fibonacci(n-1)+Fibonacci(n-2)
* #Klasa Server qe merr thread ne parameter
* class Server(threading.Thread):
* def \_\_init\_\_(self,addr,data,server): # konstruktori i klases
* threading.Thread.\_\_init\_\_(self)
* self.data = addr
* self.message = data
* self.server = server
* def run(self):
* indata = (self.message).decode()
* sdata = indata.encode()
* vinput = regex(indata)
* if vinput[0] == 'ipaddress':
* print("Received from client: ", indata)
* exe = self.data
* exe = str(exe[0]).encode()
* self.server.sendto(exe,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* elif vinput[0] == 'port':
* print("Received from client: ", indata)
* exe = self.data
* exe = str(exe[1]).encode()
* self.server.sendto(exe,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* elif(vinput[0] == 'count'):
* exe = COUNT(vinput[1])
* exe = exe.encode()
* self.server.sendto(exe,self.data)
* print ("Received from client: ", indata)
* print ("Client at " + str(self.data) + " has disconnected...")
* elif(vinput[0] == 'reverse'):
* print ("Received from client: ", indata)
* exe = REVERSE(vinput[1])
* exe = exe.encode()
* self.server.sendto(exe,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* elif(vinput[0] == 'palindrome'):
* print ("Received from client: ", indata)
* exe = Palindrome(vinput[1])
* exe = exe.encode()
* self.server.sendto(exe,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* elif(vinput[0] == 'time'):
* print ("Received from client: ", indata)
* exe = Time()
* exe = exe.encode()
* self.server.sendto(exe,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* elif(vinput[0] == 'game'):
* print ("Received from client: ", indata)
* exe = GAME()
* listToStr = ' '.join(map(str, exe)) #list to string
* listToStr = listToStr.encode()
* self.server.sendto(listToStr,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* elif(vinput[0] == 'gcf'):
* if(vinput[1].isdigit() and vinput[2].isdigit()):
* print ("Received from client: ", indata)
* exe = GCF(int(vinput[1]), int(vinput[2]))
* exes = str(exe) #int to str
* exes = exes.encode()
* self.server.sendto(exes,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* else:
* print ("Received from client: ", indata)
* errors = "Error"
* errors = errors.encode()
* self.server.sendto(errors,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* elif(vinput[0] == 'convert' and vinput[1] == 'cmtofeet'):
* if vinput[2].isdigit():
* print ("Received from client: ", indata)
* exe = cmToFeet(float(vinput[2]))
* exe = exe.encode()
* self.server.sendto(exe,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* else:
* print ("Received from client: ", indata)
* errors = "Error"
* errors = errors.encode()
* self.server.sendto(errors,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* elif(vinput[0] == 'convert' and vinput[1] == 'feettocm'):
* if vinput[2].isdigit():
* print ("Received from client: ", indata)
* exe = feetToCm(float(vinput[2]))
* exe = exe.encode()
* self.server.sendto(exe,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* else:
* print ("Received from client: ", indata)
* errors = "Error"
* errors = errors.encode()
* self.server.sendto(errors,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* elif(vinput[0] == 'convert' and vinput[1] == 'kmtomiles'):
* if vinput[2].isdigit():
* print ("Received from client: ", indata)
* exe = kmToMiles(float(vinput[2]))
* exe = exe.encode()
* self.server.sendto(exe,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* else:
* print ("Received from client: ", indata)
* errors = "Error"
* errors = errors.encode()
* self.server.sendto(errors,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* elif(vinput[0] == 'convert' and vinput[1] == 'milestokm'):
* if vinput[2].isdigit():
* print ("Received from client: ", indata)
* exe = cmToFeet(float(vinput[2]))
* exe = exe.encode()
* self.server.sendto(exe,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* else:
* print ("Received from client: ", indata)
* errors = "Error"
* errors = errors.encode()
* self.server.sendto(errors,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* else:
* self.server.sendto(sdata,self.data)
* print ("Client at " + str(self.data) + " has disconnected...")
* if \_\_name\_\_ == '\_\_main\_\_': # set variable \_\_name\_\_, and then executes all of the code found in the file.
* server = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM) # SOCK\_DGRAM, create datagram socket for UDP protocol
* server.bind(('localhost',13000))
* print ("UDP socket server is established")
* while True:
* #Recieve information from the client
* data, addr = server.recvfrom(1024)
* #Allocate a new thread to the client
* Clientthread = Server(addr,data,server)
* #Start the thread for the new client
* Clientthread.start()
* **FIEK UDP Client.py**

import socket

#Set the port

serverAddressPort   = ("localhost", 13000)

#Create a UDP socket connection

clientSocket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

#Allow the user to input a message for the server

message = input("Message to server:")

mbytes = str.encode(message)

#Send the message to the server

clientSocket.sendto(mbytes, serverAddressPort)

#Receive the reply from the server

ReceivedFromServer, serverAddress = clientSocket.recvfrom(1024) #1024bytes

srvinput = ReceivedFromServer.decode()

print("Pergjigjja:" + srvinput)

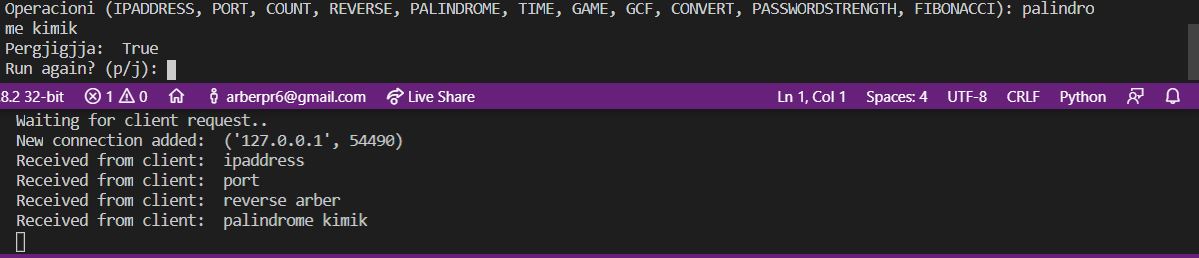
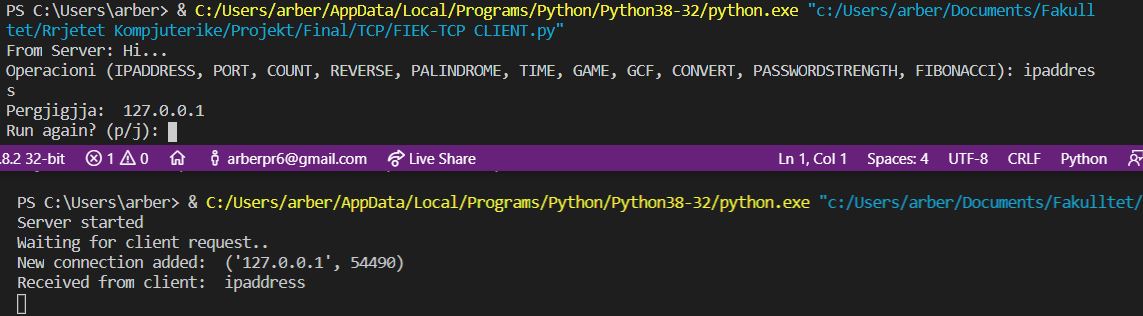
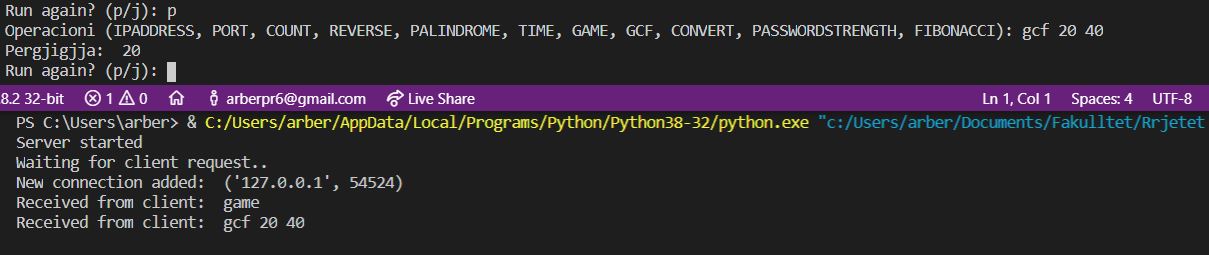
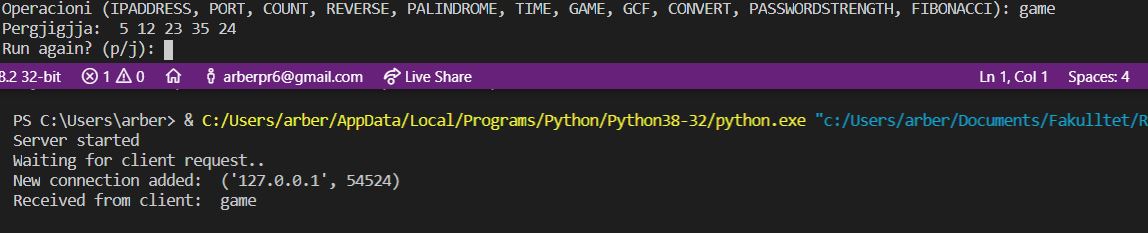
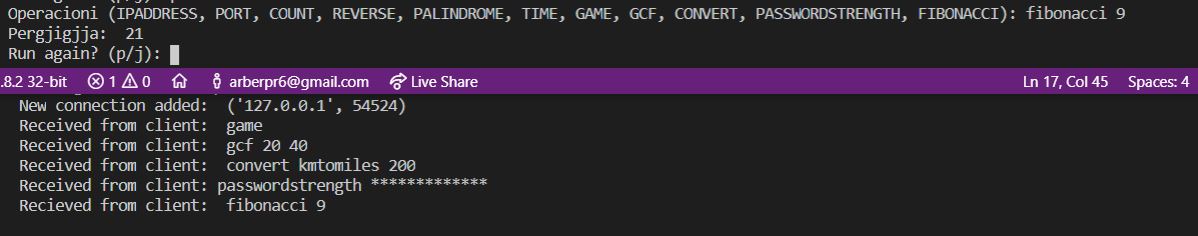
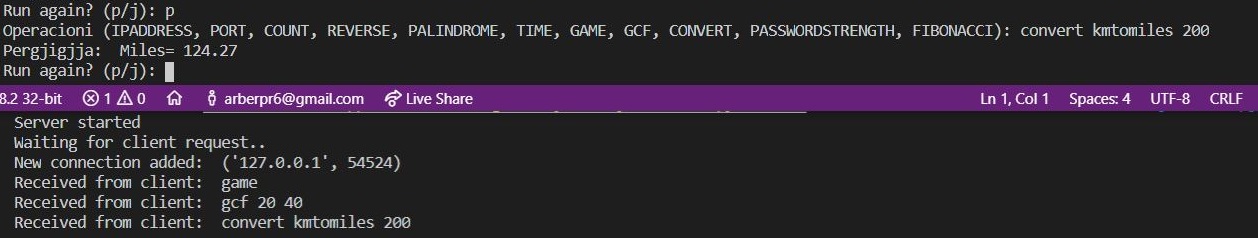
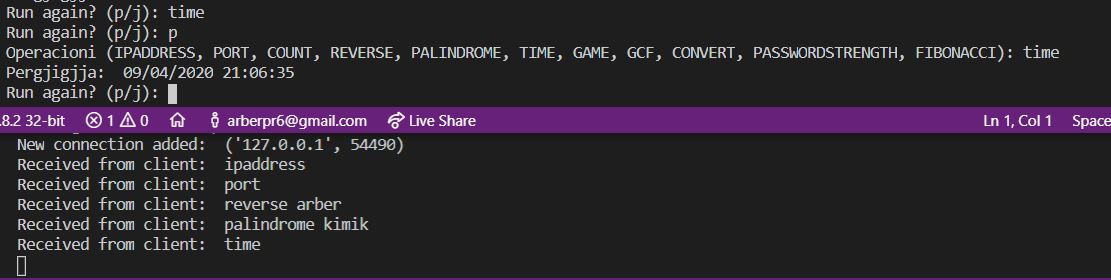
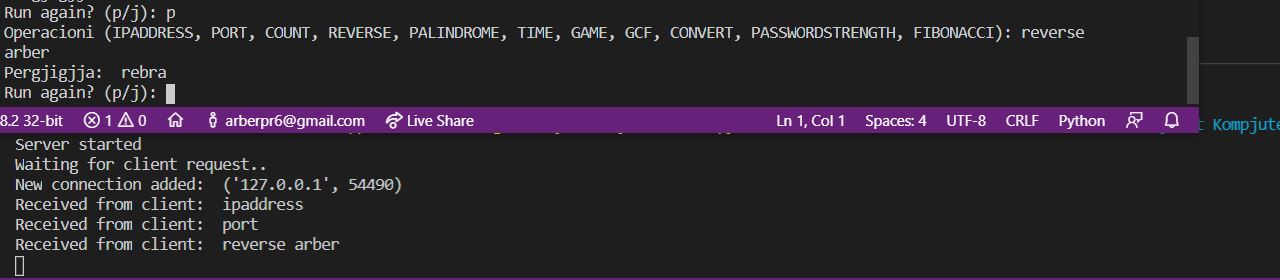
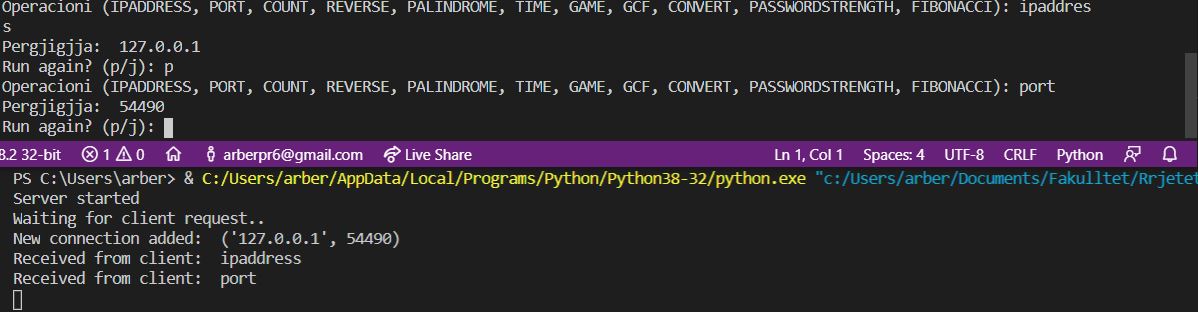
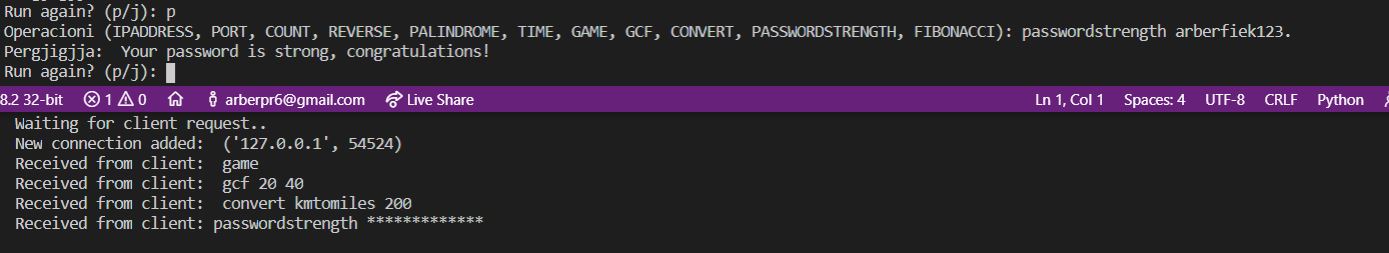
#Allow new message to be sent to the server

print("Connection closed")

#Close the client socket

clientSocket.close()

**Pamje të ekzekutimit:**

****

**Bibliografia:**

<https://docs.python.org/3/library/socket.html>

<https://www.geeksforgeeks.org/socket-programming-multi-threading-python/>

<https://dev.to/duomly/what-is-slicing-in-python-58f2>

<https://en.wikipedia.org/wiki/Fibonacci_number>

<https://docs.python.org/3/library/datetime.html>