**Universiteti i Prishtinës “Hasan Prishtina”  
  
Fakulteti i Inxhinierisë Elektrike dhe Kompjuterike**

**Programi:** Inxhinieri Kompjuterike  
 **Lënda:** Rrjetat Kompjuterike   
  
  
 **Projekti 1:** Dizajnimi Klient-Server  
  
  
  
  
Mentorët: Punoi:   
Prof. Dr. Blerim Rexha Blerta Mecini   
MSc. Haxhi Lajqi 190714100056  
  
  
  
 Skënderaj, prill 2021

Abstrakt  
  
Ky projekt është nisur nga pyetja se si mund të mundësojmë komunikimin në mes të dy aplikacioneve apo edhe të makinave të ndryshme, ku njëri dërgon kërkesa dhe tjetri pranon kërkesat dhe dërgon përgjigjie. Kjo njihet si komunikimi klient-server, dhe ky projekt paraqet pikërisht një komunikim të tillë të realizuar në gjuhën programuese Python.   
Aplikacioni i tillë që krijon këtë lidhje në mes të serverit dhe klientit është realizuar duke përdorur soketat. Në kuadër të këtij projekti përmendën konceptet themelore të protokolleve TCP dhe UDP, soketave dhe metodave të tyre. Serveri ka të implementuara disa metoda, ku për implementimin dhe thirrjën e tyre janë përdorur librari të gatshme dhe metoda të brendshme të gjuhës programuese Python.   
Zgjidhja e këtij problemi funksionon në mënyrë adekuate dhe thirrja e çdo metode jep rezultatet e pritura.

Abstract

This project is started by the question of how we can enable communication between two applications or even different machines, where one sends requests and the other receives requests and sends back responses. This is known as client-server communication, and this project represents just such communication in the Python programming language.  
Such an application that creates this connection between the server and the client is made using sockets. Within this project are mentioned the basic concepts of TCP and UDP protocols, sockets and their methods. The server has several methods implemented, where for their implementation and calling, libraries and internal methods of Python programming language are used.  
Solution of this problem works adequately and calling any method gives the expected results.

Përmbajtja

[Hyrje 3](#_Toc70478665)

[Lista e metodave të implementuara: 3](#_Toc70478666)

[Veglat e përdoruara për zhvillim dhe testim: 3](#_Toc70478667)

[Protokolli FIEK 4](#_Toc70478668)

[FIEK TCP 5](#_Toc70478669)

[- TCP serveri 5](#_Toc70478670)

[- TCP klienti 6](#_Toc70478671)

[FIEK UDP 8](#_Toc70478672)

[- UDP serveri 8](#_Toc70478673)

[- UDP klienti 8](#_Toc70478674)

[Përshkrimi dhe testimi i metodave 9](#_Toc70478675)

[Konkluzione 17](#_Toc70478676)

[Referencat 18](#_Toc70478677)

# Hyrje

Ky projekt ka për qëllim realizimin e protokolleve FIEK TCP dhe FIEK UDP , përmes të cilave mundësohet lidhja server-klient, duke përdorur gjuhën programuese Python. Qëllimet dhe objektivat e programimit të këtij projekti janë që të bëhet dizajnimi, implementimi dhe testimi i programit klient dhe server që implementohet në versionet TCP dhe UDP.  
Programimi i serverit dhe klientit është kryer duke përdorur soketat. Me rastin e krijimit të serverit, mundësohet komunikimi i tij me klientë. Serveri ka të implementuara disa metoda dhe ai punon pa ndërprerje. Serveri pret kërkesat e klientëve, ku pastaj në bazë të kërkesës që e merr nga klienti, ai ia kthen klientit përgjigjien adekuate. Serveri mund të pranojë shumë kërkesa nga i njejti klient por mund të pranojë kërkesa edhe nga shumë klientë të ndryshëm njëkohësisht dhe kjo është realizuar përmës Thread-ave. Pra, Multithreading ishte edhe pjesa më sfiduese e këtij projekti.   
Serveri është gjithmonë në dispozicion ti kthej përgjigjje klientit, edhe në rastin kur kërkesat nuk janë valide, ai do të jetë në gjendje ti kthej përgjigjie klientit duke i treguar se nuk ka dhënë kërkesë valide dhe serveri nuk do të dështojë nëse pranon kërkesa të tilla.  
Serveri i përkrah vetëm disa metoda dhe ato janë të paraqitura në listën e mëposhtme.

## Lista e metodave të implementuara:

* IP
* NRPORTIT
* KOHA
* LOJA
* PALINDROM
* NUMERO
* GCF
* ANASJELLTAS
* KONVERTO
* FIBONACCI
* THENJA
* PERFUNDO

Serveri që përkrah UDP nuk ka metodën PERFUNDO pasi që vetëm një kërkesë mund të dërgohët për datagram, mirëpo nëse klienti nuk jep ndonjë kërkesë atëherë soketi mbyllët në mënyrë automatike.

## Veglat e përdoruara për zhvillim dhe testim:

* Gjuha programuese : Python 3.6
* Visual Studio - Community Edition 2019
* Sistemi operativ : Windows 7

# Protokolli FIEK

Protokolli FIEK është një protokoll shumë i thjeshtë që i lejon klientit dhe serverit ti testoj lidhjet e tyre. Ky nuk është një protokoll standard. Është TCP versioni i cili quhet FIEK-TCP dhe UDP versioni i cili quhet FIEK-UDP. Fillimisht disa sqarime në lidhje me TCP dhe UDP protokollet, si dhe multithread-at:

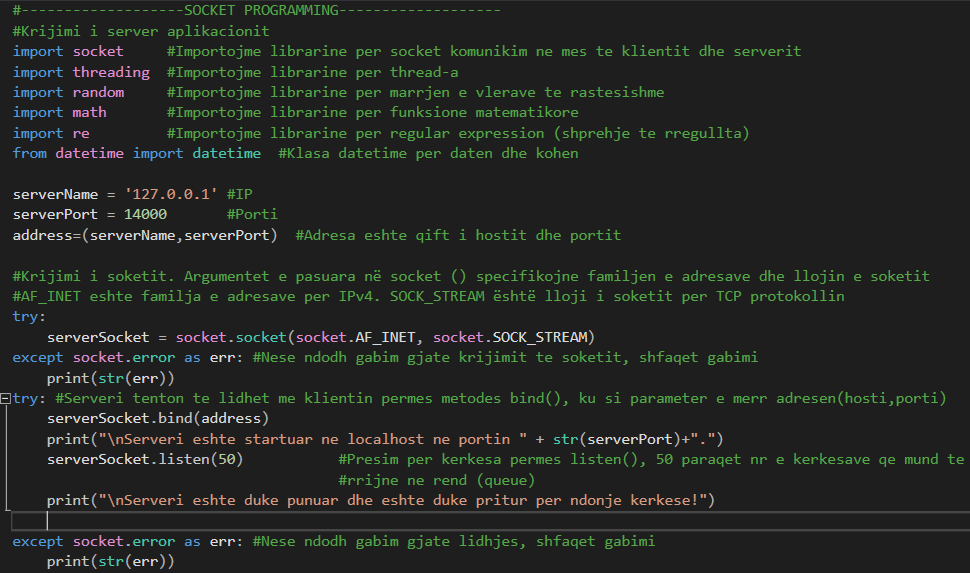
* **TCP**-Transmission Control Protocol
* **UDP-** User Datagram Protocol
* **TCP -** është protokolli i kontrollit të transmisionit, një protokoll kryesor i një koleksioni të protokolleve të internetit. Ky protokoll e plotëson Internet Protokollin (IP) dhe së bashku krijojnë koleksionin TCP/IP. TCP është një protokoll i cili është connection-oriented që do të thotë se një lidhje vendoset dhe mbahet derisa serveri dhe klienti të kenë mbaruar shkëmbimin e mesazheve mes tyre. TCP përcakton mënyrën e ndarjes së të dhënave të aplikacionit në paketa që rrjetet mund të shpërndajnë, dërgon dhe pranon paketa nga shtresa e rrjetit, menaxhon kontrollin e rrjedhës dhe për shkak se ka për qëllim të sigurojë transmetim të të dhënave pa gabime merret me ritransmetimin e paketave të dështuara ose të ngatërruara dhe njeh të gjitha paketat që mbërrijnë. TCP përdoret për organizimin e të dhënave në një mënyrë që siguron transmetimin e sigurt midis serverit dhe klientit dhe garanton integritetin e të dhënave të dërguara në rrjet, pavarësisht sasisë. Për këtë arsye, përdoret për të transmetuar të dhëna nga protokolle të tjerë të nivelit më të lartë që kërkojnë të mbërrijnë të gjitha të dhënat e transmetuara.
* **UDP -** është një nga anëtarët thelbësorë të protokolleve të Internetit. Me UDP, aplikacionet kompjuterike mund të dërgojnë mesazhe, në këtë rast të referuara si datagrame, te hostet e tjerë në një rrjet të Protokollit të Internetit (IP). UDP përdor një model të thjeshtë komunikimi pa lidhje (connectionless). Ai nuk ka handshaking dialogues, dhe kështu e ekspozon programin e përdoruesit ndaj çdo mosbesueshmërie të rrjetit themelor, dhe nuk ka asnjë garanci të dorëzimit te paketave. UDP është i përshtatshëm për qëllime ku kontrollimi dhe korrigjimi i gabimit ose nuk janë të nevojshëm ose kryhen në aplikacion. Aplikacionet e ndjeshme në kohë shpesh përdorin UDP sepse hedhja e paketave shpeshëherë preferohet më shumë, sesa të presë paketat e vonuara për shkak të ritransmetimit, gjë që mund të mos jetë një mundësi në një sistem në kohë reale.

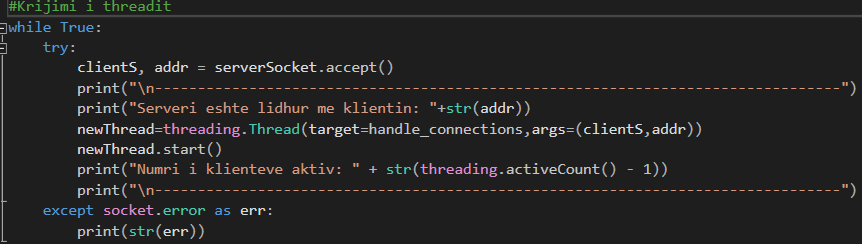
**Multithreading** - Në arkitekturën kompjuterike, multithreading është aftësia e një njësie qendrore të përpunimit CPU (ose një bërthame të vetme në një procesor multi-core) për të siguruar thread-a (fije) të shumëfishta të ekzekutimit njëkohësisht, të mbështetura nga sistemi operativ. Përderisa sistemet e multiprocesimit përfshijnë njësi të shumta të përpunimit të plotë në një ose më shumë bërthama, multithreading synon të rrisë përdorimin e një bërthame të vetme duke përdorur paralelizmin në nivelin e thread-ave, si dhe paralelizmin në nivelin e udhëzimeve. Ndërsa të dy teknikat janë plotësuese, ato kombinohen në pothuajse të gjitha arkitekturat moderne të sistemeve me CPU të shumëfishtë dhe me CPU me bërthama të shumëfishta.

## FIEK TCP

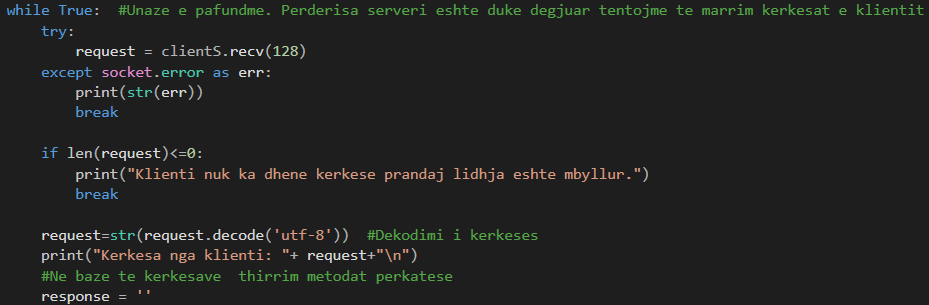
### - TCP serveri

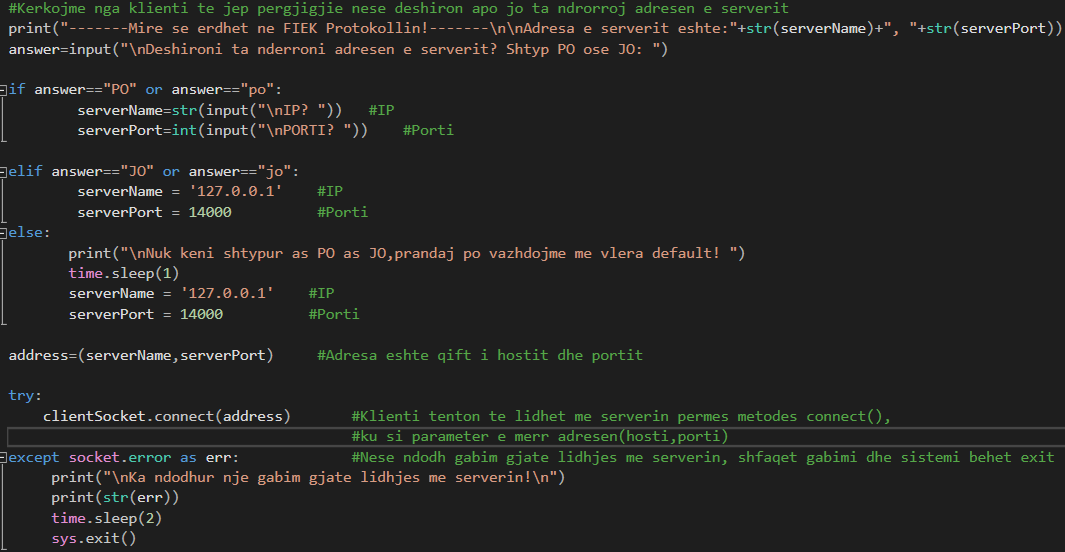
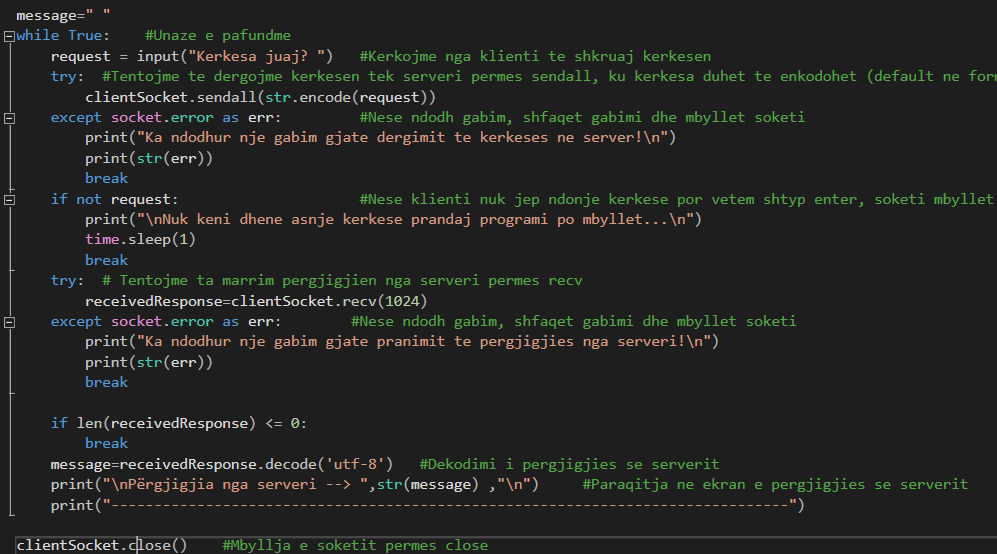
Fillimisht për të mundësuar krijimin e soketit duhet importuar librarinë **socket**. Kemi krijuar multithreaded socket server prandaj kemi importuar librarinë **threading** që e mundëson këtë gjë. Për manipulime të ndryshme gjatë implementimit të metodave importojmë libraritë **random**, **math**, **datetime** dhe **re**. Në dy variabla e ruajmë ip dhe portin e serverit. Pastaj përmës komandës **serverSocket=  
socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)** e krijojmë server soketin. Argumentet e pasuara në **socket** () specifikojnë familjen e adresave dhe llojin e soketit. **AF\_INET** është familja e adresave për **IPv4**. **SOCK\_STREAM** është lloji i soketit per TCP protokollin. Trajtimi dhe perjashtimi i gabimeve gjatë krijimit të soketit është realizuar përmes **try** **except**. Pra, nëse nuk realizohet operacioni në try, me anë të except e paraqesim gabimin që ka ndodhur. Pasi të krijohet soketi ai duhet të lidhet në atë adresë përmes metodës **bind**(), dhe pastaj pret për kërkesa të klientit përmes metodës **listen**(). Numri i pasuar te kjo metodë paraqet numrin e klientëve që mund të presin në rend për pranimin e përgjigjiës nga serveri.

  
  
Përderisa serveri është duke dëgjuar për lidhje me klient duhet ekzistuar një mënyrë për pranimin e lidhjës me klientin. Kjo mund të realizohet përmes metodës **accept**().Meqë serveri duhet të ketë mundësinë të lidhet dhe pranoj kërkesa nga klientë të ndryshëm njëkohësisht atëherë duhët përdorur thread-at. Kjiimi i thread-it bëhet përmës **threading.Thread(target,args)**, ku si parametër e merr **target** dhe **args**. Target paraqet një funksion të cilin e kemi krijuar për trajtimin e kërkesave të klientit, ndërsa args janë argumentet e atij funksioni të cilat paraqesin të dhënat e pranuara nga klienti dhe adresën e klientit, ku adresa është qift i ip dhe portit. Për startimin e tij duhet thirrur metodën **start()**. Për të kontrolluar numrin e klientëve aktiv, mund ta bëjmë duke përdorur metodën **threading.activeCount()**. Kësaj vlere që e kthen kjo metod ia zbresim 1 për arsye se një thread aktiv gjithmonë është main thread.



Kur serveri të jetë lidhur me klientin atëherë në metodën **handle\_connections(clientS,addr)** pranojmë kërkesat e tij përmes metodës **recv**(). Gjatësia maksimale e kërkesës që serveri mund të pranoj nga klienti është 128 byte. Nëse nuk ka kërkesë nga klienti atëherë lidhja me të shkëputët përmes metodës **close**(). Nëse ka kërkesa atëherë thirrët metoda përkatëse në varësi të kërkesës për të cilat do të flasim më vonë. Në një variabël e ruajmë përgjigjiën që e jep serveri për kërkesën përkatëse, dhe përmes metodës **sendall**() ku si parametër e merr përgjigjien e enkoduar, e dërgojmë përgjigjien te klienti.

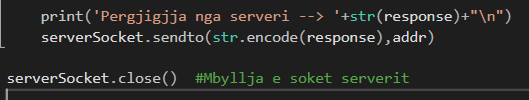
   
  
  
  
  
- TCP klienti

Fillimisht importojmë librarinë **socket**, **time** dhe **sys**. Librarinë time e importojmë për të vendosur delay dhe sys për të mbyllur programin në mënyrë eksplicite nëse nuk realizohët krijimi i klient soketit ose lidhja me serverin. Krijimi i klient soketit bëhët në mënyrë të ngjashme sikur tek serveri. Në mënyrë që klienti të mund ta ndryshoj adresën e serverit, nëse dëshiron që të lidhet me ndonjë server tjetër ndryshe nga ai që ne kemi krijuar më herët, e kemi lënë mundësinë që përmës input shfytëzuesi të shtyp PO për ta ndryshuar adresën JO për të vazhduar me vlera të nënkuptuara (default) dhe nëse përgjigjia është ndryshe nga PO ose JO të vazhdojë me vlera të nënkuptuara.   
   
  
Pastaj klienti lidhet me atë adresë përmes metodës **connect**(). Trajtimi i gabimeve dhe përjashtimet është bërë në mënyrë të ngashme me serverin. Edhe mënyra se si klienti pranon dhe dërgon të dhënat është e njejtë me serverin.  
Kur klienti lidhet me serverin atij i paraqitet lista e operacioneve që mund të kryej. Klienti e ka mundësinë të zgjedhë nga opsionet e dhëna, ndërsa nëse nuk jep ndonjë kërkesë, ose shtyp PËRFUNDO mbyllet soketi. Klienti e ka mundësinë të dërgojë më shumë së një kërkesë pasi input-in për shkruarjën e kërkesës së tij e kemi vendous brenda një unaze të pafundme, siç shihet edhe në figurën më poshtë.   
  


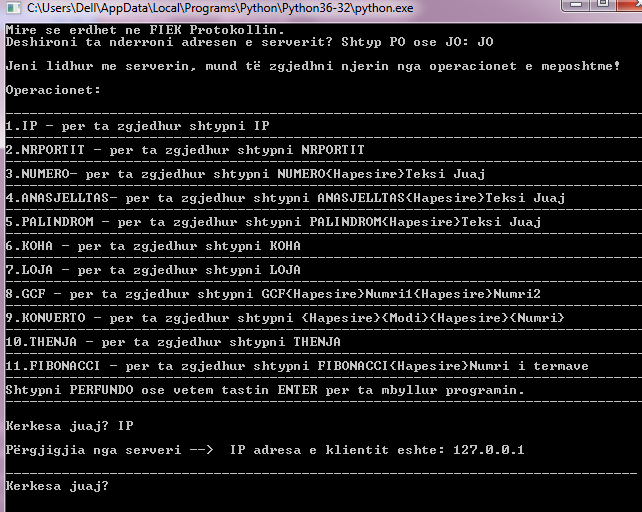
FIEK UDP

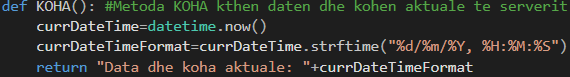
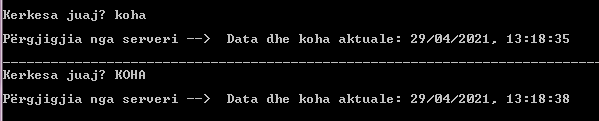
### - UDP serveri

Serveri i realizuar përmës protokollit UDP ka disa dallime të vogla por esenciale nga ai TCP . Fillimisht për krijimin e soketit në vend të **SOCK\_STREAM** përdorët **SOCK\_DGRAM** pasi që serveri UDP i dërgon përgjigjiet me datagrame. Meqenëse soketat UDP nuk janë soketa të lidhura, komunikimi bëhet duke përdorur metodat e soketave **sendto()** dhe **recvfrom()**. Këto metoda thjeshtë dërgojnë dhe marrin direkt nga një adresë e caktuar.   
Një server UDP duhet të hapë një soket dhe të marrë të dhënat që vijnë. Nuk ka **listen**() ose **accept**(). E rëndësishme të theksohet është fakti se ndryshe nga një server TCP, një server UDP mund të trajtojë shumë klientë direkt pasi nuk krijon lidhje. Mund të marrë nga çdo klient dhe të dërgojë përgjigjen. Nuk nevojiten thread-at si në serverat TCP. Po ashtu nuk është e implementuar metoda **PERFUNDO** pasi që klienti e dërgon një kërkesë për datagaram dhe serveri e dërgon vetëm një përgjigjie, prandaj nuk e kam parë të arsyeshme implementimin e saj. Kur klienti nuk jep fare kërkesë atëherë klient soketi mbyllet, dhe në server paraqitet mesazhi që klienti është shkëputur. Metodat e përdorura janë paraqitur më poshtë:

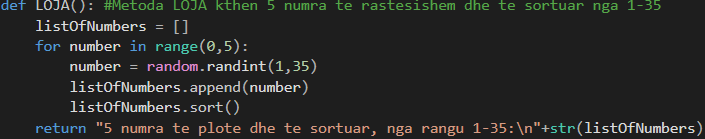
  
  
  


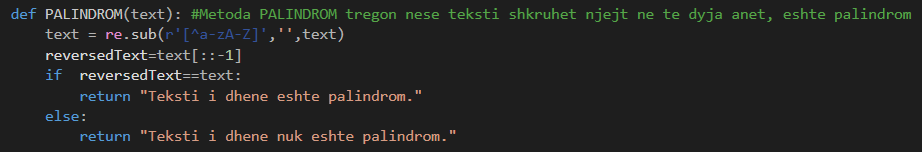
- UDP klienti   
  
Klienti i realizuar përmës protokollit UDP ka disa dallime të vogla por esenciale nga ai TCP . Fillimisht për krijimin e soketit në vend të **SOCK\_STREAM** përdorët **SOCK\_DGRAM** si tek serveri UDP. Meqenëse soketat UDP nuk janë soketa të lidhura, komunikimi bëhet duke përdorur metodat e soketave **sendto**() dhe **recvfrom**(). Këto metoda thjeshtë dërgojnë dhe marrin direkt nga një adresë e caktuar. Klienti mund të dërgojë vetëm një kërkesë për datagram.Nëse klienti nuk dërgon asnjë kërkesë soketi dhe programi mbyllen. Nëse ka gabime gjatë krijimit të soketit, dërgimit të kërkesës ose pranimit të përgjigjiës , paraqitet mesazhi që tregon qka e ka shkaktuar gabimin, soketi mbyllet dhe po ashtu edhe programi. Kjo eshte realizuar permes try except ngjashëm si tek rastët e mëhershme.

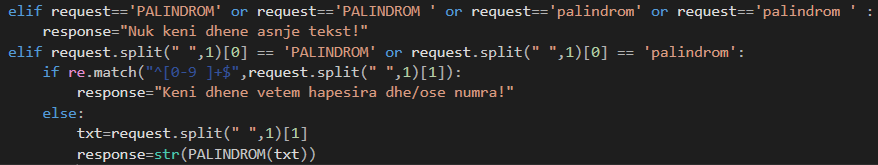
Përshkrimi dhe testimi i metodave   
  
Serveri si ai TCP ashtu edhe UDP kanë të impemetuara metodat e cekura tek hyrja. Tani do i spjegojmë me detaje  
  
**IP** – Nëse kërkesa e klientit është IP, atëherë serveri do ta kthej metodën IP, e cila paraqet një mesazh duke treguar IP adresën e klientit. Kur e marrim kërkesën e klientit p.sh. përmes **request,addr = serverSocket.recvfrom(128)** tek UDP serveri, atëherë tek request na ruhet e dhëna ndërsa tek addr na ruhet adresa e klientit që e ka bërë atë kërkesë, që është qift i IP dhe Portit. Pra meqë addr(IP,Port) , atëherë me addr[0] e marrim vetëm anëtarin e parë që paraqet IP.   
Implementimi, thirrja dhe testimi:   
  
   
  
  
  
**NRPORTIT** – Ngjashëm nëse kërkesa e klientit është NRPORTIT, atëherë serveri do ta kthej metoden NRPORTIT e cila përmes addr[1] e kthen nr e portit të klientit.

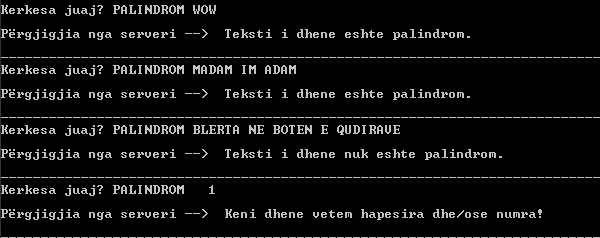
**KOHA -** Nëse kërkesa e klientit është KOHA atëherë kthehet metoda KOHA e cila e tregon datën dhe kohën aktuale të serverit. Kjo metodë shfrytëzon librarinë **datetime** si dhe për ta paraqitur në një format më të kuptueshëm për shfrytëzuesin është përdorur metoda **strftime**() për formatime të datave.   
Implementimi, thirrja dhe testimi:  
   
  
 

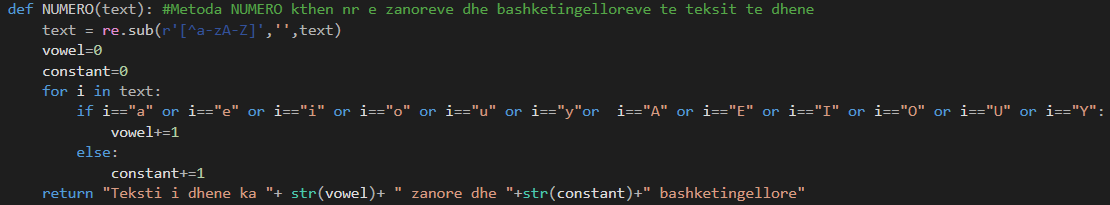
**LOJA -** Metoda LOJA kthen 5 numra të rastësishëm nga ragu 1-35, të sortuar. Kjo është realizuar duke përdorur librarinë **random** e cila mundëson që të merrën vlerat e rastësishme. Për sortim është përdorur metoda built in e vargjeve, **sort**().   
Implementimi dhe testimi:

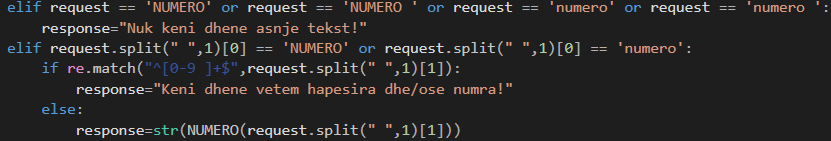
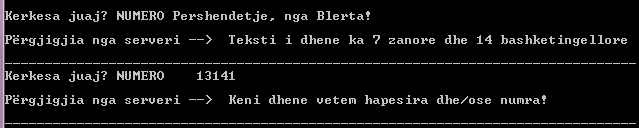
  
  
  
  
**PALINDROM** – Me anë të kësaj metode tregojmë se a është një tekst i dhënë palindrom, pra a shkruhët njësoj nga të dyja anët. Për ta realizuar këtë fillimisht nga kërkesa që vjen nga klienti përmës regular expression kemi kontrolluar që teksti të përmbajë vetëm shkronja, po ashtu të largohën hapësirat në mënyrë që të tregojmë a është palidnrom apo jo edhe për fjali jo vetëm për një fjalë. Po ashtu në mënyrë që nga kërkesa e klientit ta marrim vetëm pjesën e tekstit dhe që ta validojmë, gjatë thirrjës së metodës kam kryer manipulime të ndryshme, që po i paraqes më poshtë.   
Implementimi, thirrja dhe testimi:

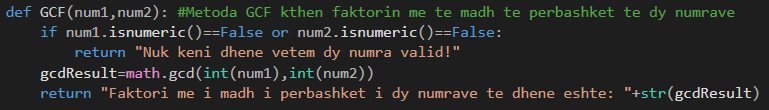
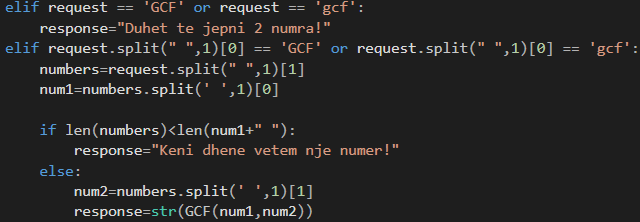
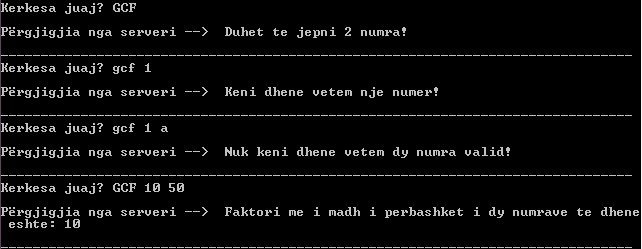


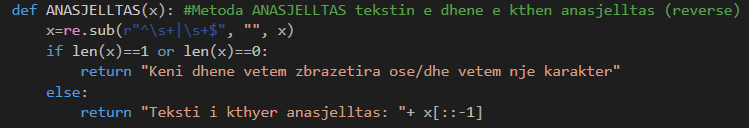


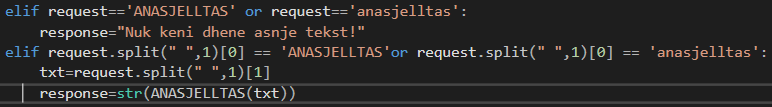
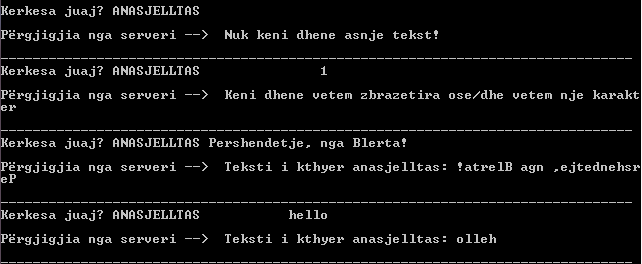
  
  
  
**NUMERO** – Kjo metodë kthen numrin e zanoreve dhe bashkëtingëlloreve në tekstin e dhënë. Për të mos i llogaritur karakteret speciale, numrat dhe hapësirat është kryer validimi përmës regular expressions. Pra në text ruhën vetëm shkonjat pasi tek r-required i kemi shkruar vetëm shkronjat a-z dhe A-Z. Pastaj kontrollojmë nëse shkronjat janë zanore dhe për çdo zanore rritet vlera e variablës vowel. Në të kundërtën rritet vlera e variablës constant, qe i numëron bashkëtingëlloret.  
Implementimi, thirrja dhe testimi:



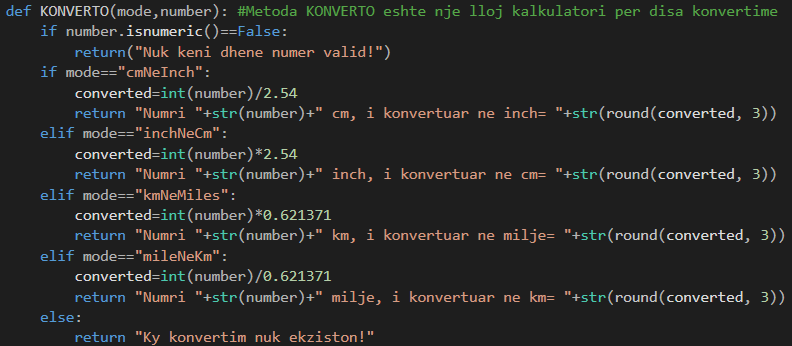
  
  
   
  
**GCF** – Kjo metodë kthen faktorin më të madhë të përbashkët të dy numrave të dhënë. Fillimisht kontrollojmë nëse parametrat e dhënë janë numra apo jo, përmes metodës **isnumeric**(). Nëse nuk janë numra shfaqet mesazhi se shfrytëzuesi nuk ka dhënë dy numra valid. Përndryshe nëse janë numra atëherë thirret metoda built in e librarisë math, **math.gcd**() që i krahason numrat dhe e kthen faktorin më të madh të përbashkët të tyre. Po ashtu gjatë thirrjës së metodës kontrollojmë që shfrytëzuesi i jep saktësisht dy numra perms kontrollimit të gjatësisë së inputit të shfrytëzuesit duke përdorur metodën **len**(). Implementimi, thirrja dhe testimi:

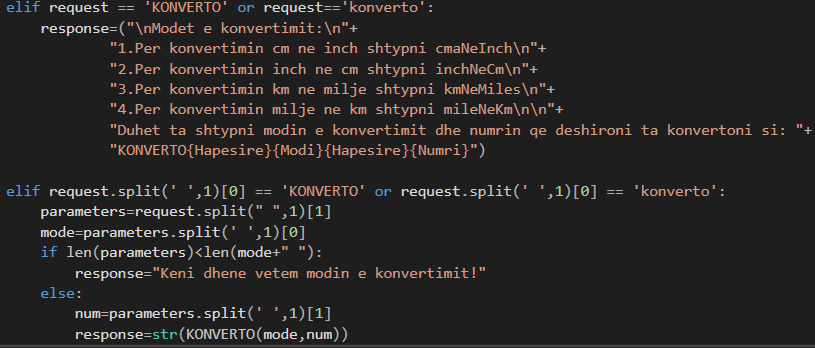
  
  
  
**ANASJELLTAS** – Kjo metodë tekstin e dhënë e kthen revers. Përmes regex i hjekim zbrazëtirat në fillim dhe në fund të tekstit. Pastaj teskti që na mbetët pa hapësira kontrollojmë që a e ka gjatësinë 1 ose zero, dhe në këtë rast nuk e kthejmë tekstin anasjelltas pasi teksti i zbrazët ose me një karakter do ishte e pakuptimtë të kthehej anasjelltas. Implementimi, thirrja dhe testimi:

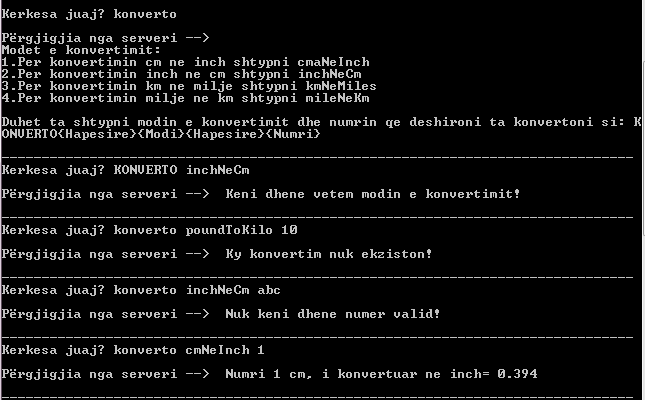


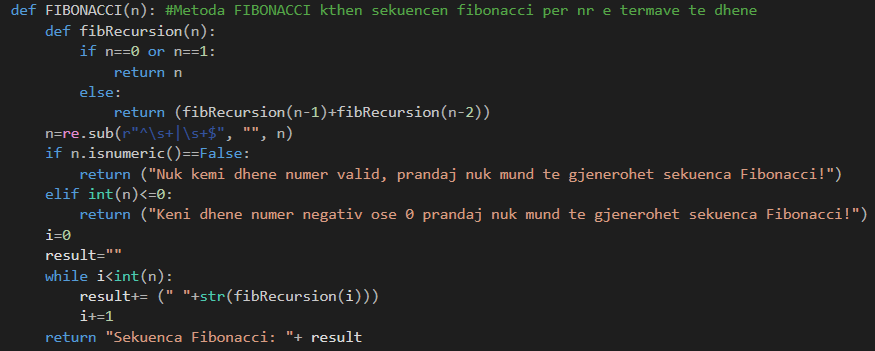
  
  


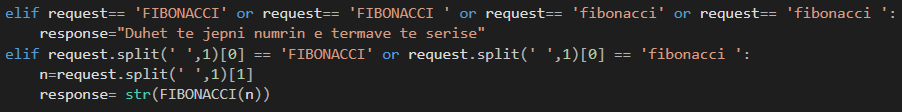
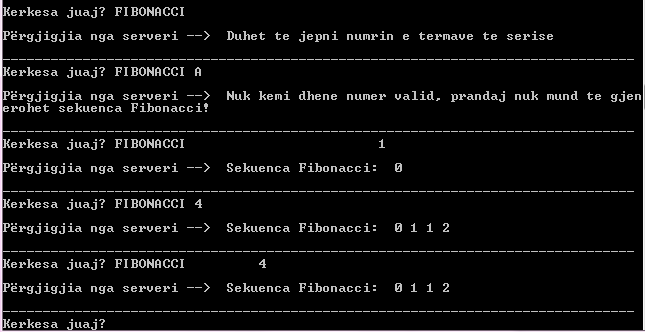
**KONVERTO** – Kjo metodë është si një lloj kalkulatori që mundëson llogaritjën e konvertimeve të ndryshme. Nëse shfrytëzuesi jep ndonjë mod të konvertimit që nuk është valid, paraqitet mesazhi që nuk është mod valid. Po ashtu është validuar edhe që parametri i dytë që paraqet numrin që do ta konvertojmë të jetë numër, jo shkronjë apo karakter special. Po ashtu rezultati i konvertimit është i rumbullaksuar përmes **round**.  
Implementimi, thirrja dhe testimi:

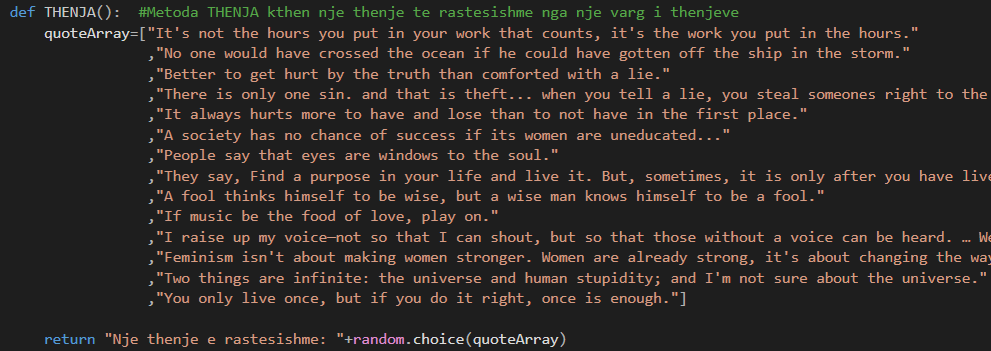
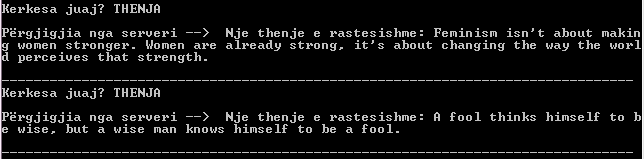




  
**FIBONACCI** – Kjo metodë kthen sekuencën Fibonacci për numrin e termave që e jep shfrytëzuesi.  
Kjo është realizuar përmes rekursionit. Nëse shfrytëzuesi në vend të numrit të termave shtyp shkronja apo karaktere speciale atëherë paraqitet mesazhi tek shfrytëzuesi dhe nuk mund të gjenerohet sekuenca. Po ashtu nëse shfrytëzuesi jep numër negative ose zero të termave, atëherë është e pakuptimtë të gjenerohët seria Fibonacci. Po ashtu përmes regular expressions numrit ia largojmë hapësirat në fund dhe në fillim, për arsye që nëse shfrytëzuesi jep numër por para dhe/ose mbara tij jep edhe hapësira, të mund të gjenerohet sekuenca . Implementimi, thirrja dhe testimi:



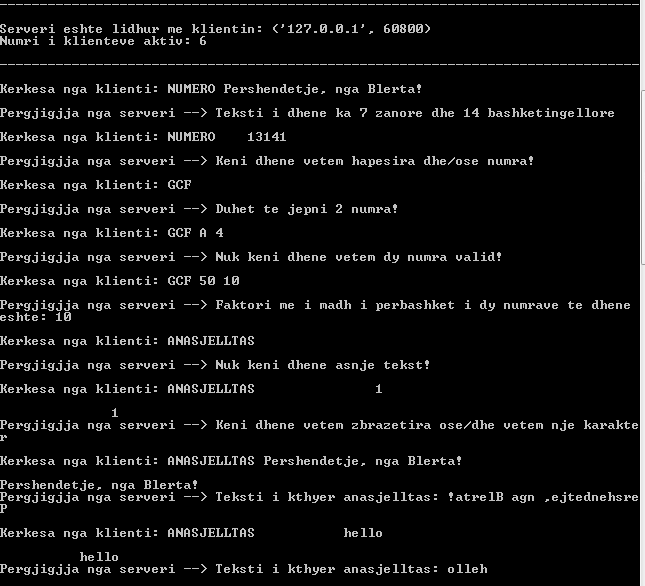
  
  


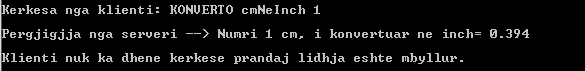
**THENJA** – Kjo metodë kthen një thënje të rastësishme nga një varg i thënjëve. Pra, fillimisht disa thënje i kemi ruajtur në një varg, Pastaj përmes metodës **random.choice ()** zgjedhet rastësisht njëri nga anëtarët e vargut. Implementimi dhe testimi :   
  
   


**PERFUNDO** – Nëse klienti shtyp PËRFUNDO atëherë paraqitet mesazh në server se lidhja me klientin e caktuar është shkëputur dhe del nga unaza e pafundme . Instruksioni i radhës është që të mbyllet soketi, prandaj soketi mbyllet dhe përfundon programi. Implementimi dhe testimi:


**Pamje të serverit gjatë pranimit të kërkesave të ndryshme:**





Konkluzione  
  
Nga rezultatet e testimeve që i kam bërë kam arritur në konkluzionin se të gjitha metodat e implementuara funksionojnë si duhet dhe kthejnë rezultatet e pritura.   
Realizimi i këtij projekti ka qenë një ndihmesë për mua në kuptimin e koncepteve themelore të programimit me soketa, multithread-ave por edhe të sintaksës së gjuhës Python.   
Me anë të realizimit të këtij projekti kam arritur në përfundimin që ky projekt mund të ndihmojë edhe të tjerët në përforëcimin e koncepteve themelore të Socket Programming , sic janë krijimi i soketave , lidhja e tyre pëmes metodës **bind**() dhe **connect**() , kuptimi i koncepteve **AF\_INET** , **SOCK\_STREAM**, **SOCK\_DGRAM** , pranimi i kërkesave me metodën **recv()** dhe **recvfrom**(), dërgimi i përgjigjjes përmes metodës **sendall()** tek protokolli TCP dhe **sendto()** tek protokolli UDP , mbyllja e soketave përmes **close()** etj. Gjithashtu , ky projekt jep një spjegim më të qartë për të kuptuar dallimet mes TCP protokollit dhe UDP protokollit .   
Gjatë tërë këtij projekti jam munduar që të gjejë zgjidhjet më adekuate të problemeve dhe ta bëjë kodin më të kuptueshëm edhe për lexuesit. Zgjidhje më efektive dhe më në hap me trendët e fundit të programimit do ishte realizimi përmes programimit të orientuar në objekte. Sidoqoftë gjithmonë ka vend për përmirësime, andaj jam e hapur për kritika, diskutime dhe bashkëpunime.

# Referencat

<https://www.geeksforgeeks.org/>

<https://www.w3schools.com/python/default.asp>

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