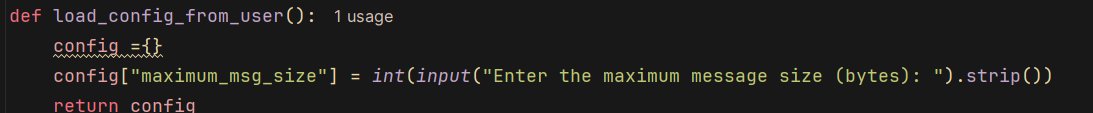
**def load\_config**

פונקציה שנמצאת גם בclient וגם server נמצאת הפונקציה הזאת שתפקידה לטפל בקריאה של הקובץ.   
כל התוכן של הtxt מועבר לדיקשינרי ולאחר מכן בפונקציה client וserver כל אחד לוקח את הערכים שרלוונטים אליו.

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי

**def load\_config\_from\_user()**



תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי

נמצאת גם בקליינט וגם בסרבר , מותאמת לפי הערכים שכל אחד אמור לקבל. בפונקציה הזאת מתבצע הזנה של input בזמן ריצה על ידי המשתמש.

**def parse\_ack**

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי

הפונקציה נועדה לנתח מחרוזת המייצגת את ה-ACKים שהתקבלו, ולחלץ ממנה את מספרי החבילות שעליהן התקבל ACK

את המחרוזת שמתקבלת מחלקים לחלקים על ידי שימוש במילה ACK כמפריד. כל חלק מכיל את הטקסט שנמצא בין מופעים של ACK.

עבור כל חלק שנמצא- מסירים רווחים מיותרים באמצעות strip, בודקים אם החלק מכיל רק ספרות באמצעות isdigit.במידה והתנאים מתקיימים מבצעים המרה של המחרוזת למספר שלם (int).

לאחר ההמרה מוסיפים את המספר לרשימת ack\_numbers שמכילה את כל מספרי החבילות עליהן התקבלה תגובה מהשרת

אם הרשימה ack\_numbers ריקה (לא נמצאו מספרים תקינים):

נזרקת שגיאה (ValueError) עם הודעה שמציינת שהפורמט של ה-ACK אינו תקין.

שגיאות כלליות- במידה ומתעוררת שגיאה אחרת, היא מודפסת למסך והפונקציה זורקת שגיאה כללית על פורמט לא תקין.

דוגמה לאופן הפעולה של הפונקציה :

קלט: "ACK123 ACK456 ACK789"

פלט: [123, 456, 789]

**def split\_message**

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי

חלוקת ההודעות- מכיוון שגודל ההודעה שהשרת יכול לקבל מוגבלת על ידי גודל ההודעה המקסימלי , נצטרך לפצל את המחרוזת שלנו לחלקים הנקבעים

דבר ראשון אנחנו מחשבים מה גודל ההודעה שאנחנו רוצים לשלוח לשרת החישוב מתבצע באמצעות 2 פעולות. אנחנו מבצעים המרה של המחרוזת לבתים (שימוש ב encode). התוצאה שנקבל היא אובייקט מסוג bytes שמייצג את המחרוזת בצורה בינארית. נשתמש בlen בשביל לדעת מה הגודל שלה בבתים .

נבצע. נבדוק – האם הגודל של ההודעה (בבתים) קטן מהגודל המקסימלי שניתן לשלוח לשרת . אם כן נשלח את כל ההודעה מבלי לפצל אותה בחבילה אחת – 0M

אם לא – נבצע פיצול של ההודעות על ידי סכימה של הגודל של ההודעה , הגודל המקסימלי פחות 1 וחילוק ללא שארית (//) במקסימום. התוספת הזו מוודאת שכל שארית תחשב כחלק נוסף וכך מובטח שהחישוב יהיה נכון גם אם יש שארית שלא מתאימה בדיוק לחלוקה.

לאחר שקבענו מה יהיה הגודל של כל חבילה , אנחנו מחזיקים רשימה ריקה

אנחנו מבצעים לולאה לפי מספר החלקים שההודעה חולקה – כלומר זה מספר החבילות שנצטרך בסופו של דבר לשלוח לשרת וכל חבילה בגודל של המקסימום אותו השרת יכול לקבל.  
בתוך הלולאה מבצעים חישוב של האינדקס התחלה ואינדקס סוף של כל חבילה בחלוקה  
start- תחילת החלק הנוכחי – כפולה של הגודל המקסימלי

end- בשביל שלא נחרוג מגבולות הגודל משמשים בmin מבין לקחת את הקטן מבין start+max לבין גודל ההודעה בבתים (message size). כאשר נבצע חלוקה למקטעים יכול להיות מקרה שבו המספר הכולל של הבתים לא מתחלק באופן שווה בגודל. במקרה כזה החלקים הראשונים יהיו בגודל של הmax והחלק האחרון יכלול רק את מה שנשאר מההודעה , שהוא יכול להיות קטן מהגודל המקסימלי לכן ככה נוודא שאם אנחנו חורגים מהגבולות כשאנחנו מוסיפים את הגודל המקסימלי סימן שגודל ההודעה הכולל קטן יותר ולכן נבחר בו  
ניצור תווית בפורמט Mi כאשר M מייצג שזאת חבילה שנשלחת מלקוח לשרת (כך הוגדר במטלה) ו-i מייצג את מספר החבילה שאותה אנחנו שולחים לשרת.

נבצע חיתוך של המחרוזת הכוללת המיוצגת בבתים לפי החבילה הנוכחית בה אנחנו כרגע מטפלים.

החיתוך יתבצע בין האינדקסים של ההתחלה והסוף שהגדרנו עבור כל חבילה. לאחר שחתכנו את הנתונים שאנחנו צריכים נבצע המרה מבתים לstring.  
עכשיו – לאחר שמתוך כלל המחרוזת ייצגנו את הקטע שאנחנו צריכים (מחרוזת) נכניס אותו לתוך הרשימה שמכילה את כלל המקטעים .  
בסופו של דבר הפונקציה תחזיר את כלל המקטעים שישלחו לשרת בהתאם להגבלות הגודל של השרת , ובהתאם לגודל של המחרוזת הנשלחת.

**def send\_data**

אנחנו קוראים לפונקציה שמפצלת את ההודעה לחבילות בהתאם לגודל המקסימלי על מנת לקבל רשימה של כל החבילות שאותם אחנו אמורים לשלוח ואז אנחנו בודקים כמה חבילות נצטרך לשלוח סך הכל.

base מייצג את המספר הסידורי של ה-ACK האחרון שהתקבל (החבילה האחרונה שאושרה).

next\_seq מייצג את המספר הסידורי של ההודעה האחרונה שנשלחה.  
אנחנו מתחילים כאשר base,next מאותחלים ל- 0 כי עדיין לא שלחנו הודעות ולכן עדיין לא קיבלנו תגובות

כל עוד המספר הסידורי של הACK האחרון (שזה בעצם החבילה האחרונה) לא מתקבל אנחנו נמשיך לשלוח חבילות תחת מגבלות החלון.

כל עוד המספר הסידורי של ההודעה האחרונה קטן מהACK האחרון + מגודל החלון – כלומר שאנחנו נמצאים בתוך גבולות חלון ההזזה וגם המספר הסידורי של ההודעה האחרונה קטן ממספר החבילות שנשלח סך הכל (כלומר אנחנו בתחום ההודעות אותם צריך לשלוח ולא חורגים ממנו) נבצע שליחה

את השליחה אנחנו מבצעים עם רווחה (\n) על מנת להשתמש בזה בשרת לזהות איפה נגמר כל מקטע ומתחיל הבא.

אנחנו יכולים לבצע שליחה של ברצף בחלון הנוכחי בלי לחכות לACK לכן כל פעם נעלה את next וכך נעקוב אחר מה היא הודעה האחרונה שנשלחה בחלון הנל

לאחר ששלחנו את כל החבילות במסגרת חלון ההזזה, אנחנו נכנסים ללולאה שבודקת אם יש ACKים שמתקבלים מהשרת. הלולאה פועלת כל עוד base קטן מ-next\_seq, כלומר, כל עוד יש חבילות שנשלחו ועדיין לא התקבל עליהן ACK.

במהלך הלולאה, אנחנו מגדירים Timeout עבור החיבור באמצעות settimeout, כך שאם לא מתקבל ACK בפרק הזמן שהוגדר, נוכל לטפל בכך.

כאשר מתקבל ACK מהשרת, אנו מפענחים את הנתונים שהתקבלו באמצעות הפונקציה parse\_ack הפונקציה מחזירה את המספרים הסידוריים של ה-ACKים שאושרו. אנו מעדכנים את base כך שיכיל את המספר הסידורי הגבוה ביותר של ה-ACKים שהתקבלו, בתוספת 1, כדי להצביע על החבילה הבאה שממתינה ל-ACK. כמו כן, אם התקבל ACK, אנו מאתחלים את הטיימר מחדש על ידי איפוס של timer\_start

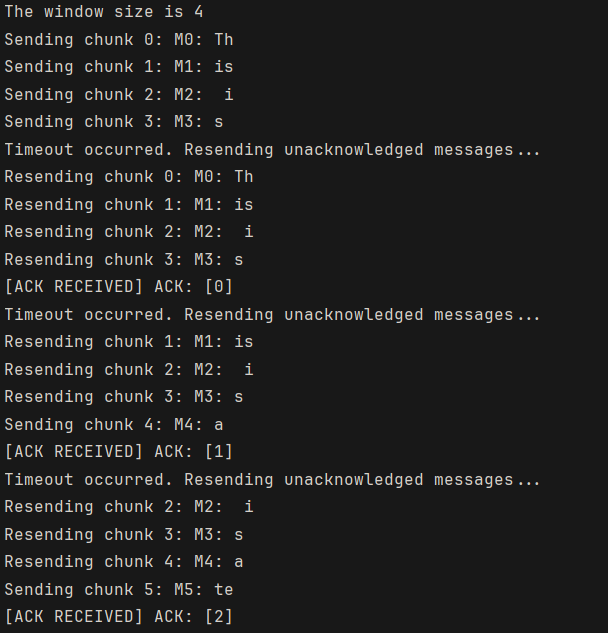
במקרה של Timeout, כלומר, אם הזמן שהוקצב לקבלת ACK חלף מבלי שהתקבלו אישורים, מתבצעת שליחה חוזרת של כל החבילות שטרם אושרו. החבילות נשלחות מהמספר הסידורי הנוכחי של base ועד next\_seq לאחר השליחה החוזרת, אנו מאתחלים את הטיימר שוב כדי להתחיל מדידה חדשה של הזמן המוקצב.

המנגנון הזה מבטיח שהחבילות ימשיכו להישלח עד שיקבלו את ה-ACKים שלהן, תוך טיפול במקרים של עיכוב או אובדן חבילות בתקשורת.

על מנת לבדוק האם החלון מתפקד כאשר מתבצע timeout ביצענו בדיקה-  
שינינו בשרת שהוא כנס לsleep ובכך ניצור את העיכוב ונבחן את ההתנהגות תחת העיכוב שיצרנו

תמונה שמכילה טקסט, גופן, צילום מסך

התיאור נוצר באופן אוטומטי



ניתן לראות שכשאר מתרחש טיימאאוט הקליינט ישלח שוב את החבילות שעדיין לא קיבל עליהן ACK  
כאשר הוא מקבל ACK הוא מזיז את החלון בהתאם לACK שהוא קיבל ושוב- שולח את ההודעות עליהן לא קיבל ACK וחוזר חלילה עד שקיבל ACK על ההודעה האחרונה

**def client**

**תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי**

הגדרת המשתנים – הודעה , גודל חלון וטיימאאוט   
בmain מתבצעת בחירה כיצד למשוך את הנתונים ובקריאה לפונקציה מועברים הנתונים בהתאם לסוג הבחירה. לכן בנקודה זאת מתבצעת הגדרה של הנתונים לפי בחירת המשתמש בתוך הלקוח.  
לאחר הטיפול בנתונים נעבור לפתיחת הסוקט  
אנחנו יוצרים חיבור בין הקליינט לסרבר. לאחר הקמת התקשורת בינהם הלקוח מבקש מהשרת למסור לו את גודל ההודעות המקסימלי שניתן לשלוח לו בכל פעם.   
השרת מחזיר תשובה עם סטרינג לכן אנחנו מבצעים הפרדה של המקסימום משאר המחרוזת באמצעות פעולות על מחרוזות ( במקרה זהז אנחנו מפצלים את מה שנמצא לפני ה " : " לבין מה שאחרי מה שנותן לנו מערך של 2 מקומות. מכיוון שמה שאחרי זה יהיה המספר בו אנחנו מעוניינים נסתכל על התא ה2 וניקח רק אותו)  
במידה ולא מוחזרת תשובה מהשרת נזרוק שגיאה  
לאחר שהמקסימום בידנו , נעביר אותו ואת שאר הנתונים אל פונקציית הטיפול בשליחת הודעות send\_data

**def process\_request**

אנחנו משתמשים בwith client\_socket בשביל שכל עוד הלקוח לא סגר את החיבור נקבל הודעות ממנו  
ניצור בהתחלה דיקשינרי ריק שבהמשך יכיל את כל החבילות שהתקבלו מהשרת  
נאתחל משתנה שעוקב אחר המספר הסידורי של החבילה הבאה שהשרת מצפה לקבל.  
ויצרנו באפר שייוצג על ידי מחרוזת ריקה שלו נוסיף כל פעם את המידע שהתקבל מהלקוח  
  
אנחנו הגדרנו את השרת לקבל את הגודל המקסימלי (שנקבע לפי הקובץ או על ידי קלט) והוספנו 1-- אנחנו מוסיפים label לכל חבילה . אם היינו כוללים אותו בחישוב החבילה אז המחרוזת לא הייתה מחולקת מלכתחילה בהתאם למגבלת השרת אלא היא הייתה מחולקת בהתאם למגבלה בהינתן הלייבל  
אם לא מתקבלת חבילה – החיבור נסגר.

כל חבילה שהשרת קיבל מתווספת אל הבאפר שהגדרנו.

לאחר שכל המידע התקבל, כל הודעה מנותחת בנפרד בעזרת החיפוש אחרי \n בבאפר – כזכור כאשר שלחנו את ההודעות השתמשנו ב\n על מנת להקל זיהוי המקטעים בשרת ועכשיו אנחנו משתמשים בזה.   
נפריד את המידע שהתקבל כך שהמידע לפני \n הוא יהיה ההודעה שתתקבל ומה שאחרי ישאר בבאפר. החלק הראשון לפני ה" : " הוא זה שמכיל את הלייבל של החבילה . נרצה לבודד רק את המספר הסידורי ולכן נבחר את המקום שאחרי הM בתא שמכיל את הלייבל וכך בודדנו את המספר הסידורי בלבד  
בנוסף מבצעים פעולת הסרת רווחים בחלק שאחרי הלייבל של החבילה (השתמשנו לשם הדפסה )  
עבור כל הודעה כזאת מחזירים לסט שאתחלנו בתחילת הפונקציה את המספר הסידורי של החבילה ובודקים-

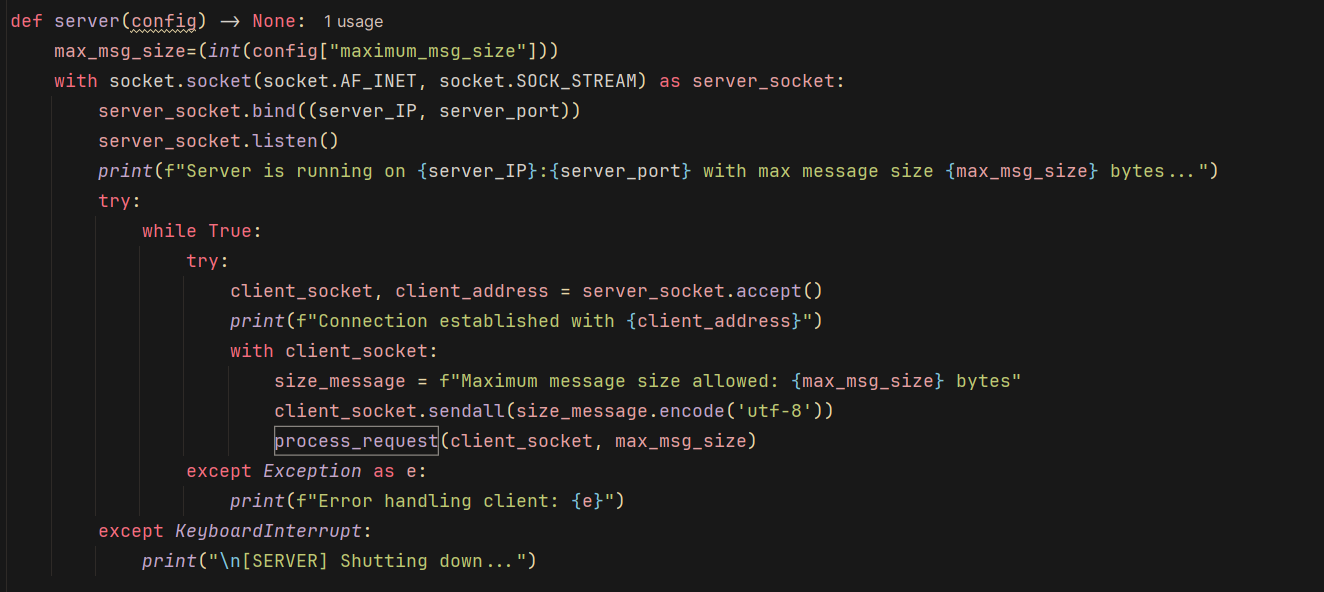
כל עוד ניתן להעלות את ה-ACK הכי גבוה שקיבלנו ב-1, ואם המספר הסידורי הזה נמצא בתוך הסט של ה-ACKים שהתקבלו, נעלה אותו.

במילים אחרות, אנחנו בודקים אם החבילה התקבלה ברצף שנשלח. אם כן, נשלח חזרה את ה-ACK החדש ונעדכן אותו ב-1.

אם לא קיבלנו את ה-ACK עבור החבילה הנוכחית, נשאיר את ה-ACK הקודם כפי שהוא ולפי זה נעדכן את הקליינט

המנגנון הזה מבטיח שאנחנו עוקבים אחרי כל ה-ACKים שהתקבלו ושומרים אותם בסט , כך שנוכל לדעת אילו חבילות אושרו ברצף.  
כך בפעם הבאה שהחבילה תתקבל נוסיף אותה לרצף ונשלח ACK בהתאם

**def server**



הגדרת המשתנים – גודל הודעה מקסימלי . הראש התוכנית הגדרנו את הIP והPORT בהם נשתמש לשרת.  
בmain מתבצעת בחירה כיצד למשוך את הנתונים ובקריאה לפונקציה מועברים הנתונים בהתאם לסוג הבחירה. לכן בנקודה זאת מתבצעת הגדרה של הנתונים לפי בחירת המשתמש והמרה של המחרוזת לint. לאחר הטיפול בנתונים נעבור לפתיחת הסוקט שלאחריו השרת נמצא במצב האזנה.  
ברגע שנוצרת תקשורת עם הלקוח -הלקוח פונה אליו בשאלה על מנת לבדוק מה הגודל המקסימלי אותו השרת יכול לקבל. השרת שולח ללקוח את הגודל  
בקשות חדשות שיתקבלו מן הלקוח יטופלו בפונקציה process\_request