



# Client – server - תרגילים

ב 2 הסעיפים הראשונים ניצור את הלקוח והשרת נבדוק אותם

זה מול זה כשנסיים את כתיבת שניהם

## 1. לקוח – בחלק זה ניצור את הלקוח

- פתחו קובץ פייתון
  - אתחלו את main
- ```
if __name__ == '__main__':
    main()
```
- בראש הקובץ - יבאו את הספרייה socket
- ```
import socket
```
- הגדירו את ה IP אותו קיבלתם ואת ה PORT כקבועים
  - ב main:
  - צרו socket חדש
  - התחברו עם ה socket ל server
- ```
my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

my_socket.connect((IP, PORT))
```
- שלחו ל server את שמכם
  - קבלו תשובה מה server
- ```
my_socket.send('Homer Simpson'.encode())

data = my_socket.recv(MSG_LEN)
```
- הדפיסו את התשובה פעמיים, פעם כ byte array ופעם כ string
- ```
print("print bytes: ", data) # print bytes
print("print string: ", data.decode()) # print string
```
- ולא לשכוח.....
- ```
my_socket.close()
```

## 2. שרת – הפעם נכתוב את השרת ונריץ את השרת והלקוח שלנו

- פתחו קובץ פיתון
- אתחלו את main

```
if __name__ == '__main__':
    main()
```

- יבאו את הספרייה socket

```
Import socket
```

- ב main:

- צרו socket חדש

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

- קשרו את ה socket ל ip ו port

```
server_socket.bind((IP, PORT))
```

- האזינו על ה socket לניסיונות התקשרות

```
server_socket.listen(1)
```

- קבלו התקשרות של לקוח

```
client_socket, address = server_socket.accept()
```

- קבלו הודעה מהלקוח

```
data = client_socket.recv(MSG_LEN)
```

- הדרישה בתרגיל היא להחזיר ללקוח את המחרוזת שקיבלנו ממנו. לכן נשלח את data ל-socket של הלקוח – *client\_socket*.

```
client_socket.send(data)
```

מדוע לא צריך הפעם להמיר ממחרוזת למערך בתים?

- ולסיום לא נשכח לסגור את 2 ה sockets

```
client_socket.close()
```

```
server_socket.close()
```

- הריצו את השרת ומולו הריצו את הלקוח שלכם

### 3. ניהול exceptions

בכל פעולה על socket יכולות לקרות שגיאות. לא נרצה שהתוכנית שלנו תקרוס. את הטיפול בשגיאות נממש בעזרת exceptions. הפונקציות בספרייה socket זורקות exceptions מסוג socket.error נוסף לקוד שלנו except | try על מנת לתפוס אותן.

להזכירכם טיפול ב exceptions:

```
try:
    commands handling sockets
except socket.error as msg:
    print "socket error: ", msg
except Exception as msg:
    print "general error: ", ms
```

### 4. Server forever

עד עכשיו יצרנו שרת שמקבל פקודה אחת ויוצא ולקוח ששולח פקודה אחת ויוצא. נרצה שהשרת שלנו יריץ יותר מפקודה אחת והלקוח יוכל לשלוח יותר מפקודה אחת לכן נכניס את הקוד ללולאה.

#### השרת

נבנה שרת שיוצא רק כשהלקוח שולח לו exit (באותיות גדולות או קטנות כרצונו). נחלק את הקוד של השרת ל 3 פעולות:

- **initiate\_server\_socket(ip, port)** – פעולה המקבלת ip | port יוצרת את ה socket ומתחילה להאזין

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((SERVER_IP, PORT))
server_socket.listen(1)
return server_socket
```

- **handle\_clients(server\_socket)** – פעולה המקבלת את ה socket של ה שרת ומחכה ללקוחות (באמצעות הפעולה accept) – ברגע שמתקבלת בקשת שירות מלקוח נקראת הפעולה handle\_single\_client שמטפלת בלקוח עד שהוא מבקש לצאת או מתנתק

```
client_socket, address = server_socket.accept()
done = handle_single_client(client_socket)
```


ואז כמובן נחזור לחכות

כשנחזור מהפונקציה המטפלת בלקוח כמובן לא נשכח.....

```
client_socket.close()
```

הפעולה handle\_single\_client תחזיר True אם הלקוח שלח EXIT.  
אם הפעולה מחזירה True צאו מהלולאה

- נממש את הפעולה המטפלת בלקוח עד שהוא מתנתק  
- **handle\_single\_client(client\_socket)**

 request = client\_socket.recv(MSG\_LEN)  
client\_socket.send(request)

את השורות האלה נכניס ללולאה ממנה נצא כשנקבל EXIT או כשהלקוח התנתק:  
while request!= " and request!= 'EXIT':

לא נשכח להפוך את ה request לפני ההשוואה ממערך בתים ל מחרוזת  
החזירו מהפעולה True אם הלקוח EXIT I False אחרת.

- ולא נשכח בסוף התכנית

server\_socket.close()

שימו לב, כרגע כשהתכנית שלכם עובדת בדקו חריגות:

מה קורה כשהלקוח מתנתק?

מה קורה כשהשרת מתנתק?

כשהלקוח מתנתק על השרת לחכות ללקוח חדש, בדיוק כמו ש google אינו מפסיק לעבוד  
כשהמחשב שלכם מתנתק

עטפו את קטעי הקוד השונים ב try I except ובדקו שניתן להעלות ולהוריד את הלקוח בלי  
לאתחל את השרת

### הלקוח

נבנה לקוח שיוצא רק כשהמשתמש מקיש את הפקודה exit (באותיות גדולות או קטנות כרצונו)  
ניצור 2 פעולות:

1. **initate\_client\_socket(ip, port)** – אילו שורות קוד נכניס לתוכה? מה תחזיר הפעולה?

2. **handle\_user\_input(my\_socket)** - פעולה שבלולאה :

i. קולטת מהמשתמש מחרוזת בעזרת פקודת input. – שימו לב כדי לעבור את ה

test האוטומטי השמשו ב – **input("please enter a request")**

שימו לב לרווח

ii. שולחת אותה לשרת

iii. מדפיסה את התשובה בבתים וכמחרוזת

אם התקבל exit שולחת לשרת מדפיסה את התשובה וחוזרת

כשהמשתמש לוחץ exit נצא מהפעולה ששולחת את המחרוזות לשרת בלולאה ולא נשכח.....

הוסיפו גם בלקוח טיפול ב exceptions כך שאם השרת נופל תודפס הודעה והלקוח יצא בצורה  
מסודרת

## 5. שרת למספר פקודות – תרגיל 2.6 מהספר

בתרגילים הקודמים התבקשתם לכתוב שרתים ולקוחות שמקבלים הודעות בגודל קבוע חלק מהאתגר בתרגיל זה הינו כתיבת פרוטוקול למשלוח הודעות בין השרת והלקוח.

### שלב 1

עליכם לכתוב מערכת שרת-לקוח, כאשר השרת מבצע פקודות שהלקוח שולח אליו, ומחזיר ללקוח תשובה בהתאם. על כל בקשה של הלקוח להיות באורך של ארבעה בתים בדיוק. אורך התגובה יכול להיות שונה בהתאם לבקשה.

להלן רשימת הבקשות שיש לתמוך בהן:

- **TIME** - בקשת הזמן הנוכחי. על השרת להגיב עם מחרוזת שכוללת את השעה הנוכחית אצלו.

o היעזרו במודול datetime שמובנה בפייתון. חפשו ב google: python get current

.time

- **NAME** - בקשת שם השרת. על השרת להגיב עם מחרוזת שמייצגת את שמו. השם יכול להיות כל מחרוזת שתבחרו.

- **RAND** - בקשת מספר רנדומלי. על השרת להגיב עם מספר רנדומלי שנע בין הערכים 1 ל-10.

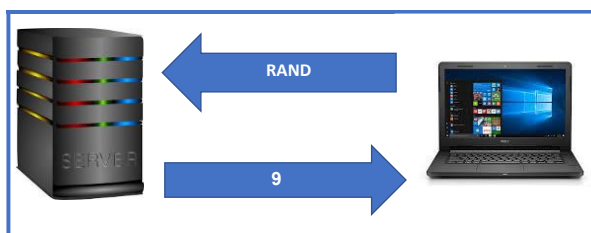
o היעזרו במודול random המובנה בפייתון. חפשו ב google: python generate random

number

- **QUIT** - בקשת ניתוק. על השרת לנתק את החיבור עם הלקוח ולחכות לבקשת חיבור מלקוח אחר. גם הלקוח עצמו יוצא

- **EXIT** - בקשת ניתוק. על השרת לנתק את החיבור עם הלקוח ולרדת בעצמו. גם הלקוח עצמו יוצא

### דוגמאות לתקשורת:



### הדרכה בעמודים הבאים....

## השרת

נשתמש בשרת ובלקוח מהתרגיל הקודם אולם הפעם נפרק את הלולאה של `handle_single_client` למספר פעולות על מנת לבצע את הבקשות המתקבלות ולענות

```
while not done:
    request = receive_client_request(client_socket)
    response = handle_client_request(request)
    send_response_to_client(response, client_socket)

    done = request == 'EXIT' or request == "QUIT"
```

`receive_client_request` – פעולה הקוראת את הפקודות מה `socket` באמצעות הפקודה `recv` מחזירה את הבקשה כמחרוזת

`handle_client_request` – פעולה המבצעת את בקשת הלקוח ומכינה מחרוזת לתשובה. **פעולה זאת תזמן פעולה עבור כל בקשה.** כלומר נכתוב את הפעולות `QUIT()` ו `TIME()`, `NAME()`, `RAND()`, `EXIT()`. הפעולות `QUIT` ו `EXIT` יחזירו את המחרוזת `exit` או `quit`. כל פעולה תחזיר את התשובה כמחרוזת שאותה נשלח לשרת ב `send_response_to_client`

`send_response_to_client` – פעולה השולחת את מחרוזת המוכנה ללקוח כ `binary array` כמובן

## הלקוח

גם בלקוח נחלק את `handle_user_input` לפעולות

```
while request.upper() != 'QUIT' and request.ipper() != 'EXIT' :
    request = input('please enter a request ')
    request = request.upper()

    if valid_request(request):
        send_request_to_server(my_socket, request)
        handle_server_response(my_socket)
    else:
        print("illegal request")
```

שימו לב לרווח

`valid_request` – פעולה המוודאת שקיבלנו פקודה חוקית כלומר `TIME`, `RAND`, `NAME` או `EXIT`

`send_request_to_server` – פעולה השולחת את הבקשה לשרת

`handle_server_response` – פעולה המקבלת את תשובת השרת ומדפיסה אותה

החזירו בבקשה `illegal request` והשתמשו ב `'please enter a request '` ה-`test` האוטומטי בודק את הפלט, (כולל הבקשה בפקודה `input`)

שימו לב לרווח

## שלב 2

## תרגול כתיבת פרוטוקול

- האם בשלב 1 הלקוח שלכם ביצע `socket.recv(1024)`? כעת, צרו פרוטוקול שמאפשר לשלוח מהשרת ללקוח הודעות באורך שונה. יכולת זו תשמש אתכם בתרגילים בהם לא תוכלו להניח שאורך ההודעה מהשרת ללקוח הוא 1024 בתים או מספר קבוע כלשהו - לדוגמה בהעברת קובץ. אפשרויות לדוגמא:
- השרת ישלח ללקוח הודעה שכתוב בה מה אורך המידע שעליו לקבל בתור תשובה.
  - השרת ישלח ללקוח הודעה מיוחדת שמשמעותה "שליחת המידע הסתיימה".

אנחנו נבחר בפרוטוקול בו השרת והלקוח ישלחו הודעה שמורכבת מ 2 חלקים

1 – 4 בתים המכילים את אורך המחרוזת - כדי לרפד מספר באפסים ניתן להשתמש בפונקציה

`zfill` של `str`

2 – המחרוזת עצמה

לדוגמא ההודעה `hello` תישלח כך:

```
encoded_msg = msg.encode()
l = len(encoded_msg)
ll = str(l)
lll = ll.zfill(4)
llll = lll.encode()
```

חישוב האורך:

0	0	0	5
---	---	---	---

```
my_socket.send(llll + encoded_msg)
```

0	0	0	5	H	e	l	l	o
---	---	---	---	---	---	---	---	---

כשיקראו הודעה יקראו אותה ב 2 חלקים:

```
raw_size = my_socket.recv(4) →
data_size = raw_size.decode()
```

0	0	0	5
---	---	---	---

```
if data_size.isdigit():
    data = my_socket.recv(int(data_size))
    return data.decode()
```

H	e	l	l	o
---	---	---	---	---

פתחו קובץ פיתון חדש בשם `protocol.py` וכתבו בו ד פרוצדורות

1. `send(sock, data)` - פעולה המקבלת `socket` ומחרוזת `data` ושולחת על פי הפרוטוקול את `data` – (כשגודלה ב 4 הבתים הראשונים בהודעה)
2. `recv(sock)` - פעולה המקבלת `socket` וקוראת מה `socket` 4 בתים שמכילים את גודל ההודעה ואחריהם את ההודעה

נשים לב כי אומנם השולח כותב את גודל ההודעה ושולח את כולה אבל שכבת התעבורה לא מתחייבת להעביר את ההודעה כיחידה אחת. לכן נקרא את ההודעה בלולאה. אם אנחנו רוצים לקרוא size בתים מ socket בשם s ניצור לולאה:

```
tot_data = b''
while size > 0:
    data = s.recv(size)
    size -= len(data)
    tot_data += data
```

שפרו את הפרוצדורה recv בקובץ protocol כך שתכיל 2 לולאות

- לולאה שקוראת את 4 הבתים הראשונים שמכילים את גודל ההודעה - size
- לולאה שקוראת מה socket עד שמגיעים כל התווים שנשלחו. כלומר עד שהגודל של tot\_data הוא size

### מה נשנה בשרת? ובלקוח?

ראשית, כדי שנוכל להשתמש בפעולות שכתבנו בקובץ protocol ניבא אותו – import protocol  
שנית, נשנה בשרת ובלקוח את הפונקציות השולחות והמקבלות הודעות כך שיבצעו עבורנו את בניית ההודעות ושליחתן ואת קבלת ההודעות ב 2 החלקים.  
מה נעשה בפעולת הללו?

- נחליף כל זימון של sock.send(data) ל protocol.send(sock, data)
- נחליף כל זימון של sock.recv(1024) ל protocol.recv(sock) את 4 הבתים המכילים את size - הגודל של ההודעה ואחריהם size בתים המכילים את ההודעה

### אילו פעולות נשנה?

בשרת:

- receive\_client\_request
- send\_response\_to\_client

בלקוח:

- send\_request\_to\_server
- handle\_server\_response



## שלב 3

## הפרדת הפעולות

צרו קובץ חדש בשם methods.py והעבירו אליו את הפעולות TIME, RAND, NAME, QUIT, EXIT. העבירו לקובץ את הספריות שהפעולות משתמשות בהן (למשל rand) ואת הקבועים אם יש כאלה יבאו את הקובץ הזה - import methods.

```
import math
print(getattr(math, "pow")(2,3))
```

נשים לב ששמות הפעולות בקובץ הם בעצם הבקשות בדקו מה עושה הקוד שבמסגרת:

מה עושה הפעולה getattr? הפעולה מחזירה לנו ערכי תכונות של עצמים. בפיתון מודול הוא עצם והפעולות הן תכונות שלו. כלומר:

```
getattr(module, "method")(parameters)
```

מזמן את הפעולה method מהמודול module עם הפרמטרים parameters

יבאו את הקובץ methods לשרת – import methods  
השתמשו ב getattr בפעולה handle\_client\_request על מנת לטפל בבקשה שלנו

```
def handle_client_request(request):
    return getattr(methods, request)()
```

## שלב 4

כיוון שאנו עוסקים בהגנת סייבר, עלינו לכתוב שרת יציב, כלומר גם אם הלקוח שולח "זבל", לשרת שלנו אסור לקרוס. בדקו את יציבות השרת באמצעות הודעות מסוגים שונים והוסיפו בדיקת שגיאות וניהול Exceptions  
אם הלקוח יורד השרת ממשיך כמובן לעבוד ומחכה ללקוח נוסף. אם השרת יורד הלקוח מדפיס הודעה ויורד בצורה מסודרת

## הדרכה:

**השרת** עיטפו את זימון הפעולות מ methods ב exception מסוג Exception אם קיבלתם בקשה לא חוקית ה exception יתפוס אותה. החזירו **illegal command** במקרה הזה (ה test האוטומטי בודק את התשובה)

עיטפו את הטיפול בלקוח ב exception מסוג socket.error כך שאם הלקוח מתנתק ניתפוס את ה-exception ונעבור ללקוח הבא,

**הלקוח** עיטפו את הטיפול בלקוח ב exception מסוג socket.error. כך שאם יהיה ניתוק מהשרת הלקוח יצא

- בדקו שכשהלקוח יורד השרת ממשיך ומוכן לקבל את הלקוח הבא.
- בדקו שכשהשרת יורד הלקוח יוצא בצורה מסודרת עם הדפסה מתאימה
- מה קורה אם השרת מקבל בקשה לא חוקית

## בהצלחה