תרגיל 4.3

כתבו מהו התוכן ואילו שדות Header מתקבלים בתגובה לבקשות GET לכתובות הבאות:

1. http://http-demo.appspot.com/a
2. http://http-demo.appspot.com/x
3. http://http-demo.appspot.com/2
4. http://http-demo.appspot.com/3
5. http://http-demo.appspot.com/4
6. http://http-demo.appspot.com/something

פרוטוקול HTTP - תכנות צד שרת בפייתון, הגשת קבצים בתגובה לבקשת GET

אם ניזכר לרגע בפרק תכנות Sockets/ שרת-לקוח, נבין ששני סוגים של רכיבים משתתפים בכל תקשורת כזו - צד השרת, שמספק גישה למשאבים (נניח, מיילים), וצד הלקוח, שהוא למעשה אפליקציית הקצה שבה אנחנו משתמשים, ומבקש משאבים מן השרת. בדוגמה שראינו זה עתה, המשאבים היו דפי אינטרנט טקסטואליים, אותם סיפק השרת בתגובה לבקשה של הלקוח (הדפדפן או הכלי **telnet**). למעשה, השרת עונה לבקשות שמגיעות מהרבה לקוחות, ובינתיים אנחנו מפשטים ומדברים על לקוח בודד. נדבר בהמשך על ההשלכות של "טיפול" בלקוחות רבים.

מעט רקע היסטורי - בתחילת דרכה של השימוש הביתי ברשת האינטרנט, בסביבות שנות ה-90', שרתי אינטרנט שימשו בעיקר ל"הגשה" של עמודי תוכן סטטיים. הכוונה ב"הגשה" היא להעביר את התוכן של הקובץ שנמצא בכתובת שביקשה האפליקציה. למשל, בדוגמה שראינו בחלק הקודם, השרת הגיש לדפדפן קובץ טקסט.

נתעכב לרגע כדי להבין איך בנויות הכתובות הללו: לכל אתר ברשת יש כתובת ייחודית, שמכונה URL - ראשי תיבות של Universal Resource Locator (בעברית: "כתובת משאב אוניברסלית"). למעשה, ה"כתובות" שהכנסנו לדפדפן בסעיף הקודם היו URLים. על משמעות המילה "משאב" נרחיב עוד בהמשך.

URL בסיסי בנוי באופן הבא:

במקרה הזה, השרת שמאחורי הכתובת www.site.com יחפש בתיקייה folder_a תיקייה בשם folder_b, ובתוכה יחפש קובץ בשם file_name.txt. הפניה מתבצעת בסכמה של HTTP[22].

אם אכן קיים קובץ כזה, השרת יגיש אותו בתור תגובה לבקשה.

כדי להבין טוב יותר את הרעיון, תממשו כעת בעצמכם שרת כזה, שמגיש קבצים בתגובה לבקשות.

אל דאגה, התרגיל מפורט, ומפורק למשימות קטנות מאד.

לפני כן, נסביר מעט יותר על המימוש של **root directory**. כפי שציינו קודם, הפניה למשאב מתבצעת כמו במערכת קבצים – פנייה ל-"/folder_a/folder_b/file_name.txt" למעשה פונה לקובץ file_name.txt בתיקייה folder_b שבתיקייה folder_a. אך היכן נמצאת תיקייה folder_a? היא נמצאת בתיקיית השורש, root directory, של האתר. בפועל, התיקייה הזו היא פשוט תיקייה כלשהי במחשב שהמתכנת הגדיר אותה בתור root directory. בחירה נפוצה היא להגדיר את C:\wwwroot בתור ה-root directory. כעת, כאשר הלקוח פונה ושולח בקשה מסוג:

GET /folder_a/folder_b/file_name.txt HTTP/1.1

הבקשה תתבצע למעשה לקובץ הבא אצל השרת[23]:

C:\wwwroot\folder_a\folder_b\file_name.txt

תרגיל 4.4 – כתיבת שרת HTTP

לאחר כל אחד מהשלבים הבאים, הקפידו לבדוק את השרת שלכם על ידי הרצה של התכנית, ושימוש בדפדפן כלקוח; היזכרו במשמעות של הכתובת 127.0.0.1 אותה הזכרנו בפרק תכנות ב-Sockets - הכתובת שאליה נתחבר באמצעות הדפדפן תהיה http://127.0.0.1:80 (כאשר 80 הוא הפורט בו נשתמש). על מנת לבדוק את הפתרון שלכם, אנו ממליצים להוריד אתר לדוגמה מהכתובת:www.cyber.org.il/networks/webroot.zip . העתיקו את תוכן קובץ ה-ZIP אל ספריה כלשהי (כמובן שיש לפתוח את הקובץ) והשתמשו בה בתור ה- root

[22] ברוב המקרים בסכמה יצוין פרוטוקול - כגון HTTP או FTP. עם זאת, במקרים מסוימים, היא לא תכלול פרוטוקול, כמו הסכמה "file".

[23] זאת בהנחה שהשרת מריץ מערכת הפעלה Windows. כאמור, במערכות הפעלה שונות ה-path עשוי להיראות בצורה שונה.

directory שלכם. המטרה היא שהשרת ישלח ללקוח את index.html ויתמוך באפשרויות השונות שיש בעמוד אינטרנט זה.

1) כתבו שרת המחכה לתקשורת מהלקוח בפרוטוקול TCP בפורט 80. לאחר סגירת החיבור על ידי הלקוח, התוכנית נסגרת.

2) הוסיפו תמיכה בחיבורים עוקבים של לקוחות. כלומר, לאחר שהחיבור מול לקוח נסגר, השרת יוכל לקבל חיבור חדש מלקוח.

3) גרמו לשרת לוודא כי הפקטה שהוא מקבל היא HTTP GET, כלומר - ההודעה שהתקבלה היא מחרוזת מהצורה שראינו עד כה: מתחילה במילה GET, רווח, URL כלשהו, רווח, גירסת הפרוטוקול (HTTP/1.1), ולבסוף התווים \r ו- \n.

- אם הפקטה שהתקבלה אינה HTTP GET - סגרו את החיבור.

4) בהנחה שהשרת מקבל בקשת HTTP GET תקינה ובה שם קובץ, החזירו את שם הקובץ המבוקש אל הלקוח. בשלב זה, החזירו את שם הקובץ בלבד, ולא את התוכן שלו.

- שימו לב שב-Windows משתמשים ב-"\" כמפריד בציון מיקום קובץ, בעוד שבאינטרנט וגם בלינוקס משתמשים ב-"/".
- הערה: את שם הקובץ יש להעביר בתור שם משאב מבוקש, ולא ב-Header נפרד.
- הערה נוספת: בשלב זה, אל תעבירו Headerים של HTTP כגון הגירסה או קוד התגובה.

5) כעת החזירו את הקובץ עצמו (כלומר, את התוכן שלו).

- אם מתבקש קובץ שלא קיים - פשוט סגרו את החיבור (היעזרו ב- os.path.isfile).
- הערה: בניגוד לכמה מהתרגילים הקודמים, כאן יש לשלוח את כל הקובץ מיד, ולא לחלק אותו למקטעים בגודל קבוע (כפי שעשינו, למשל, בתרגיל 2.7).

6) הוסיפו את שורת התגובה ו-Headerים של HTTP:

- גרסה HTTP 1.0.
- קוד תגובה: 200 (OK).
- השורה Content-Length: (מלאו בה את גודל הקובץ שמוחזר).

7) במקרה שבו לא קיים קובץ בשם שהתקבל בבקשה, החזירו קוד תגובה 404 (Not Found).

8) אם השרת מקבל בקשת GET ל-root (כלומר למיקום "/") - החזירו את הקובץ index.html (כמובן, וודאו שקיים קובץ כזה; תוכלו ליצור קובץ בשם index.html שמכיל מחרוזת קצרה, רק לשם הבדיקה).

9) אם השרת מקבל בקשות לקבצים מסוגים שונים, הוסיפו ל-Header של התשובה את השדה Content Type, בהתאם לסוג הקובץ שהתבקש. תוכלו להעזר בנתונים הבאים:

- קבצים בסיומת txt או html:

Content-Type: text/html; charset=utf-8

- קבצים בסיומת jpg:

Content-Type: image/jpeg

- קבצים בסיומת js:

Content-Type: text/javascript; charset=UTF-8

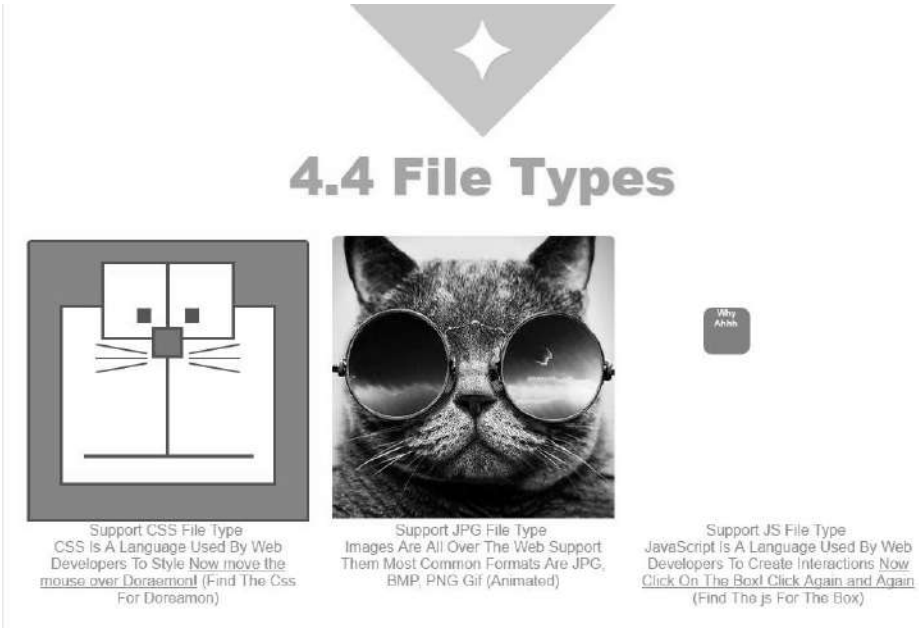
- קבצים בסיומת css:

Content-Type: text/css

10) כעת הוסיפו תמיכה במספר Status Codes נוספים (היעזרו בויקיפדיה):

1. 403 Forbidden – הוסיפו מספר קבצים שאליהם למשתמש אין הרשאה לגשת.
2. 302 Moved Temporarily – הוסיפו מספר קבצים שהמיקום שלהם "זז". כך למשל, משתמש שיבקש את המשאב page1.html, יקבל תשובת 302 שתגרום לו לפנות אל המשאב page2.html. לאחר מכן, הלקוח יבקש את המשאב page2.html, ועבורו יקבל תשובת 200 OK.
3. 500 Internal Server Error – במקרה שהשרת קיבל בקשה שהוא לא מבין, במקום לסגור את החיבור, החזירו קוד תגובה 500.

נסו את השרת שלכם באמצעות הדפדפן- גירמו לשרת לשלוח את המידע שנמצא ב-webroot ותוכלו לצפות באתר הבא:



*אתר בדיקת תרגיל שרת HTTP– קרדיט **תומר טלגם***

מדריך לכתיבה ודיבוג של תרגיל כתיבת שרת HTTP

קוד של שרת HTTP הוא קוד מורכב יחסית לתרגילים קודמים ולכן מומלץ לתכנן אותו מראש ולחלק אותו לפונקציות. יש דרכים רבות לכתוב את קוד השרת, תוכלו להתבסס על שלד התוכנית הבא:

http://www.cyber.org.il/networks/HTTP_server_shell.py

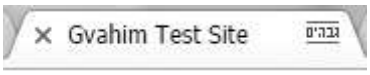
המקומות שעליכם להשלים בעצמכם נמצאים תחת הערה "TO DO". שימו לב שכדי שהתוכנית תעבוד תצטרכו להוסיף לה קבועים ופונקציות נוספות, השלד אמור רק לסייע לכם להתמקד.

<u>טיפים לכתיבה</u>

1. שימו לב שהתשובות שאתם מחזירים הם לפי כל השדות של HTTP. אל תפספסו אף סימן רווח או ירידת שורה...
2. לעיתים עלול להיות מצב שבו הן השרת והן הדפדפן מצפים לקבל מידע. כדי לצאת ממצב זה, ניתן להגדיר SOCKET_TIMEOUT. שימו לב שאם הזמן שהגדרתם עבר, תקבלו exception. עליכם לטפל בו כדי שהשרת לא יסיים את הריצה.

<u>אייקון</u>

מרבית אתרי האינטרנט כוללים אייקון מעוצב שמופיע בלשונית של הדפדפן. גם הדפדפן שלכם יבקש את האייקון מהשרת. כדי למצוא היכן נמצא האייקון, תוכלו לפתוח את העמוד index.html בתוכנת notepad++ ולחפש את favicon.ico.



רוצים ליצור לעצמכם אייקון אישי? תוכלו להשתמש באתר http://www.favicon.cc/

תוכלו להעלות לאתר תמונה כלשהי או להמציא אייקון משלכם. לאחר שסיימתם לעצב אייקון, לחצו על כפתור download icon ושימרו אותו במקום המתאים.

<u>כלי דיבוג – breakpoints</u>

שימוש ב-breakpoints לטובת דיבוג קוד השרת שלכם הוא הכרחי. צרו breakpoints במיקום שבו הקוד עדיין מבוצע באופן תקין ועיקבו אחרי הערכים שמקבלים משתנים ושפונקציות מחזירות כדי לבדוק בעיות בקוד שלכם.

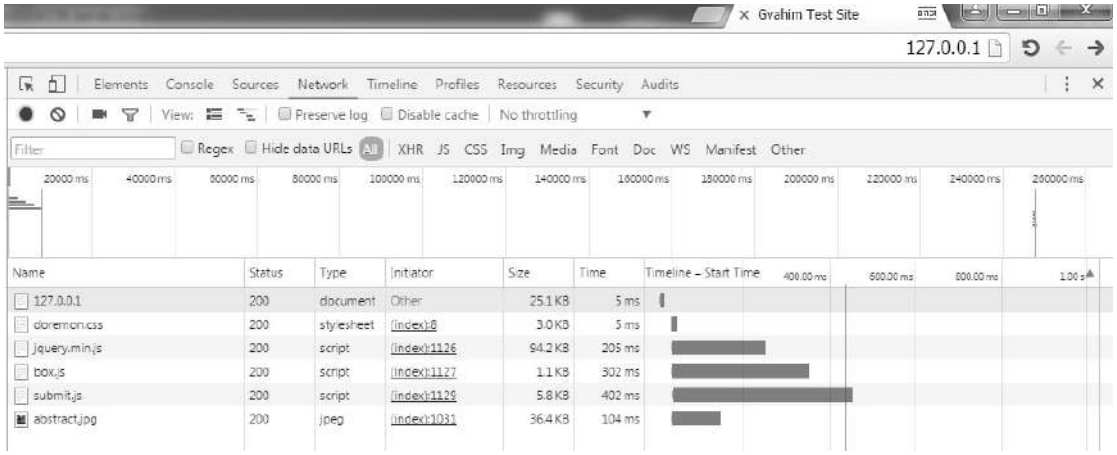
באמצעות בדיקת ערכי משתנים תוכלו לבדוק, לדוגמה, מה הבקשה ששלח הלקוח.

```
203 def main():
204     # Open a socket and loop forever while waiting for clients
205     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
206     server_socket.bind((IP, PORT))
207     server_socket.listen(10)
208     print "Listening for connection on port %d" % PORT
209
210     while True:
211         client_socket, client_address = server_socket.accept()
212         print 'New connection received'
213         client_socket.settimeout(SOCKET_TIMEOUT)
214         handle_client(client_socket)
215
216
217 if __name__ == "__main__":
218     # Call the main handler function
219     main()
220
```

<u>כלי דיבוג- דפדפן כרום</u>

דפדפן כרום כולל אפשרות לעקוב אחרי כל התעבורה בין הדפדפן לשרת. איך עושים את זה?

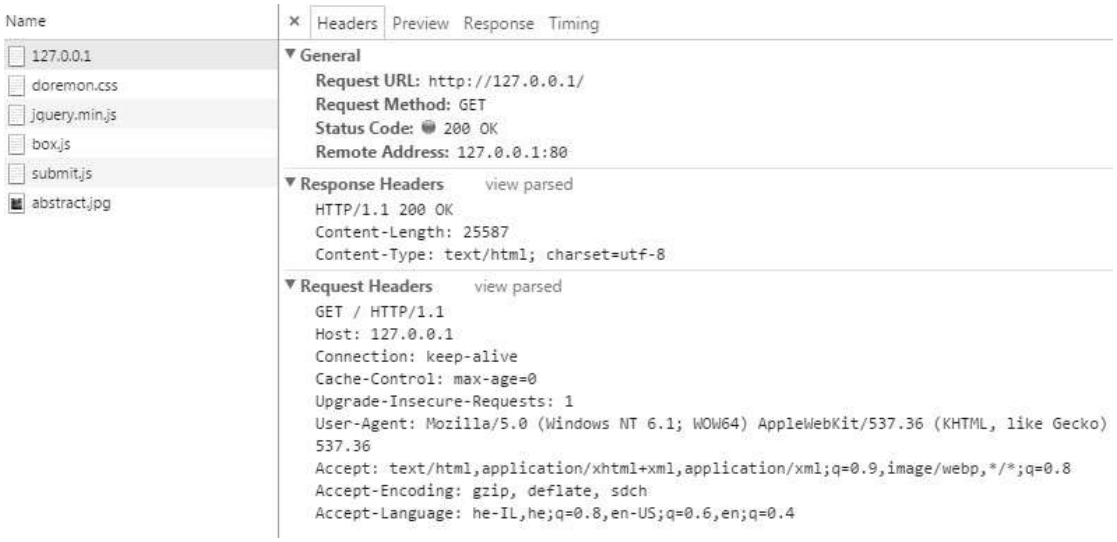
בתוך הדפדפן לחצו על F12 ולאחר מכן על טאב network. ייפתח לכם המסך הבא:



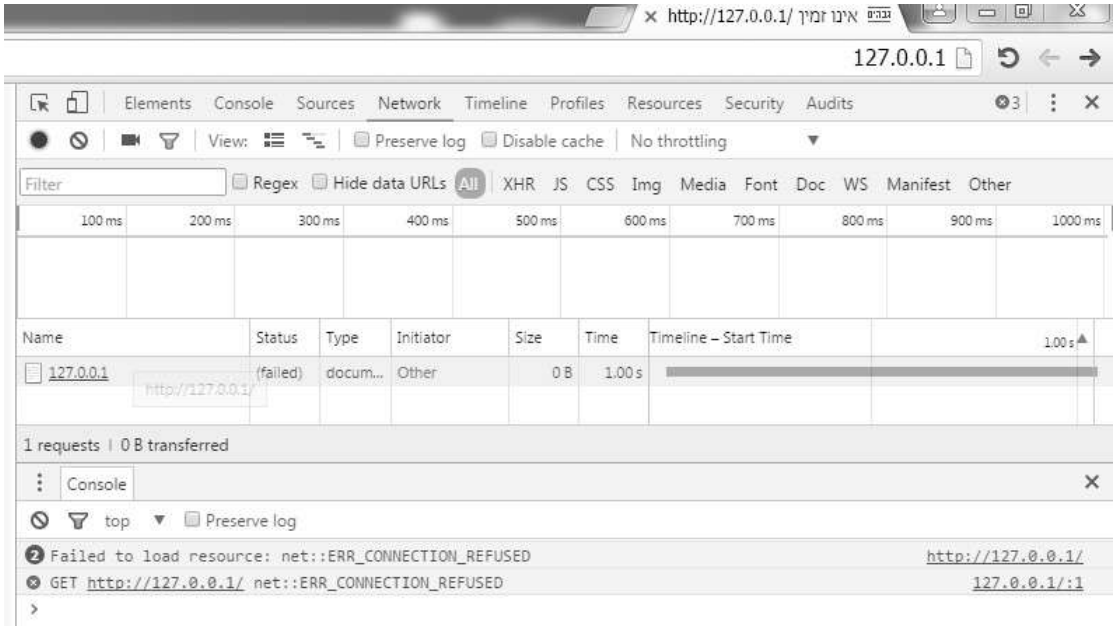
לחצו על הנקודה האדומה כדי להתחיל הקלטה חדשה.

אם תעמדו על שם של קובץ כלשהו תוכלו לקבל פרטים נוספים בטאבים החדשים שמופיעים לצידו- Headers, Preview, Response, Timing.

המידע המעניין ביותר לטובת דיבוג השרת שלכם מופיע ב-headers, שם ניתן לראות את הבקשות שנשלחו לשרת ואת התשובות של השרת.



אם מסיבה כלשהי לא התקבלו מהשרת כל הקבצים, תוכלו לראות מה לא התקבל באזור מיוחד שמוקצה לתיעוד שגיאות. לדוגמה, כאשר השרת לא מקבל את בקשת ההתחברות של הלקוח:



<u>כלי דיבוג- Wireshark</u>

כלי נפלא זה, איתו עשינו היכרות בעבר, יכול לספק לכם את כל המידע שעבר בין השרת ללקוח שלכם- בתנאי שתריצו את השרת ואת הלקוח על שני מחשבים נפרדים, שמחוברים ע"י ראוטר או switch כפי שוודאי יש לכם בבית.

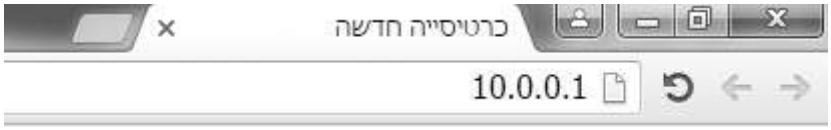
בשלב ראשון, וודאו שהשרת שלכם מאזין לכתובת IP 0.0.0.0 (ולא 127.0.0.1). כתובת 0.0.0.0 אומרת למערכת ההפעלה שלכם לשלוח לשרת שבניתם לא רק פקטות שמגיעות מתוך המחשב עצמו (127.0.0.1) אלא גם פקטות שמגיעות ממחשבים אחרים- בתנאי שהם פונים לפורט הנכון.

לאחר מכן בידקו באמצעות ipconfig מה כתובת ה-IP של השרת שלכם (שימו לב, כתובת IP ברשת הפנימית שלכם, בהמשך נלמד מה היא כתובת פנימית). בדוגמה הבאה הכתובת היא 10.0.0.1:

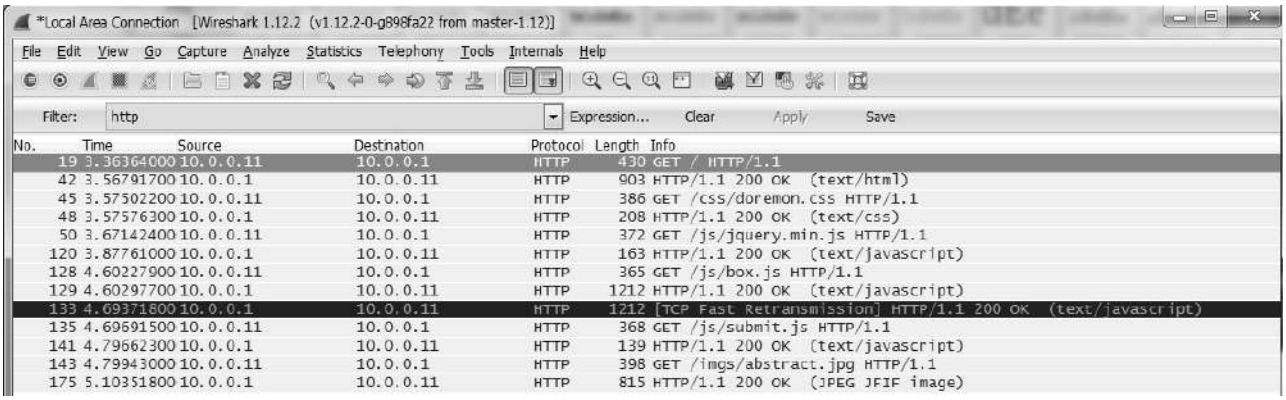
```
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.
C:\Users\ADMIN>ipconfig
Windows IP Configuration

Ethernet adapter Local Area Connection:
   Connection-specific DNS Suffix  . : Home
   Link-local IPv6 Address . . . . . : fe80::48ac:d008:caaf:7415%12
   IPv4 Address. . . . . . . . . . . : 10.0.0.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.0.0.138
```

לפני שנתקדם לשלב הבא, הפעילו wireshark.

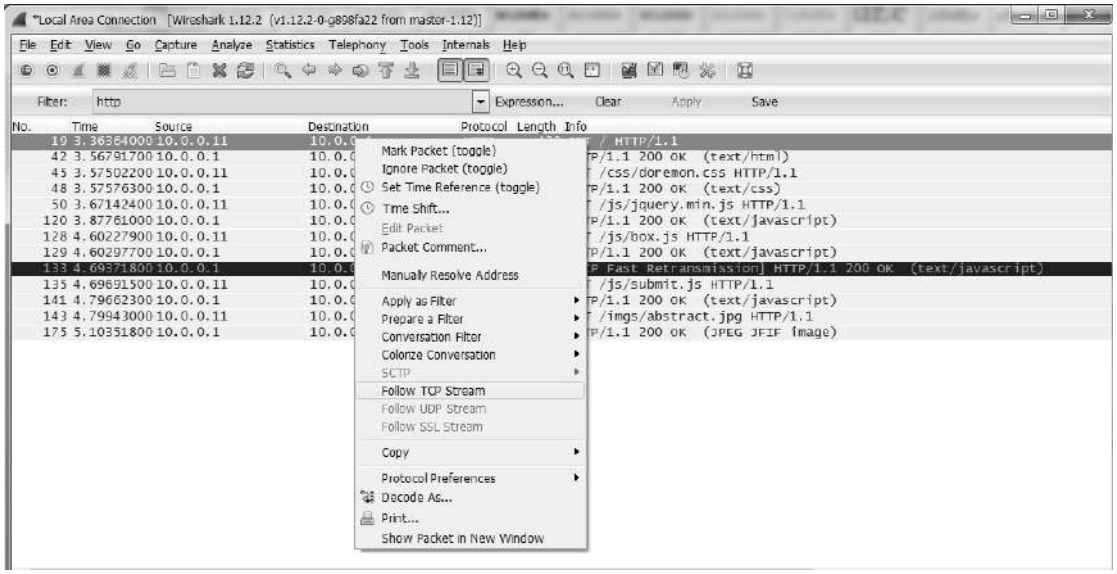
כעת עליכם לקבוע בהתאם את כתובת ה-IP שהלקוח פונה אליה:



תוכלו למצוא בתוך wireshark את התעבורה בין הדפדפן בלקוח לבין שרת ה-HTTP שלכם. נפלטר את התעבורה לפי http:

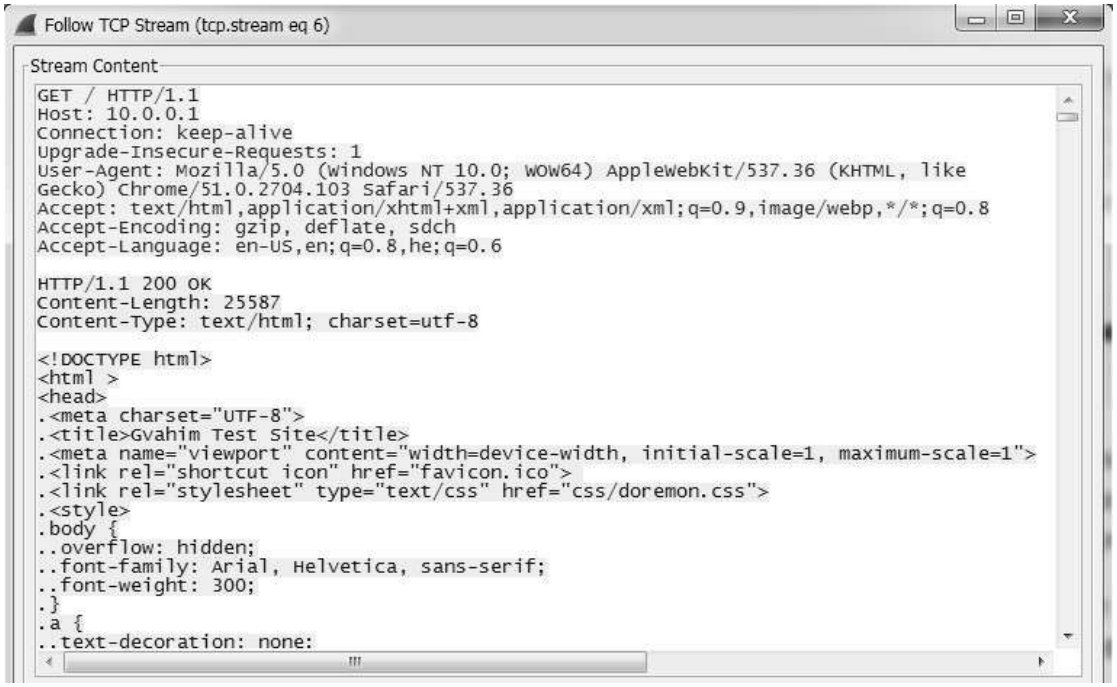


נוכל לראות את כל הבקשות של הלקוח ואת תשובות השרת. בצילום מסך זה ניתן לראות שחלק מהמידע (השורה הצבועה על ידי wireshark בשחור) שודר פעמיים מהשרת ללקוח- Retransmission- שמעיד על כך שהשרת לא קיבל אישור על המידע ששלח ללקוח (האישור הוא ברמת שכבת התעבורה, פרוטוקול TCP, עליו נלמד בהמשך). אם נרצה לעקוב אחרי כלל הפקטות שעברו בין השרת והלקוח שלנו, כולל הודעות הקמת וסיום קשר ושליחה מחודשת של פקטות, נוכל להקליק על שורה כלשהי ולבחור follow TCP stream.



יוצגו בפנינו מסכים שמראים את כל המידע שעבר, הן בשכבת האפליקציה (השרת והלקוח בצבעים שונים) והן בשכבת התעבורה.

נוכל להשתמש במידע שעבר ברשת כדי לאתר בעיות שאולי גרמו לכך שהשרת שלנו לא עובד כפי שתיכננו.



Filter: tcp.stream eq 6 Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
15	3.17640900	10.0.0.11	10.0.0.1	TCP	62	50279→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
17	3.36110200	10.0.0.1	10.0.0.11	TCP	62	80→50279 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 SACK_PERM
18	3.36343700	10.0.0.11	10.0.0.1	TCP	60	50279→80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
19	3.36364000	10.0.0.11	10.0.0.1	HTTP	430	GET / HTTP/1.1
20	3.36426100	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
21	3.36426500	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
22	3.37566600	10.0.0.11	10.0.0.1	TCP	60	50279→80 [ACK] Seq=377 Ack=1461 Win=64240 Len=0
23	3.37570300	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
24	3.37571000	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
25	3.46497200	10.0.0.11	10.0.0.1	TCP	60	50279→80 [ACK] Seq=377 Ack=5841 Win=64240 Len=0
26	3.46501300	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
27	3.46502100	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
28	3.46502400	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
29	3.46502700	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
30	3.46503000	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
31	3.46503600	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
32	3.46768100	10.0.0.11	10.0.0.1	TCP	60	50279→80 [ACK] Seq=377 Ack=7301 Win=64240 Len=0
33	3.46771500	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
34	3.46772200	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
35	3.47272600	10.0.0.11	10.0.0.1	TCP	60	50279→80 [ACK] Seq=377 Ack=10221 Win=64240 Len=0
36	3.47276100	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
37	3.47277000	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
38	3.47277300	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
39	3.47277500	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
40	3.56783700	10.0.0.11	10.0.0.1	TCP	60	50279→80 [ACK] Seq=377 Ack=14601 Win=64240 Len=0
41	3.56790800	10.0.0.1	10.0.0.11	TCP	1514	[TCP segment of a reassembled PDU]
42	3.56791700	10.0.0.1	10.0.0.11	HTTP	903	HTTP/1.1 200 OK (text/html)

... זהו, בהצלחה!

פרוטוקול HTTP - תשובה "תכנותית" לבקשות GET, ופרמטרים של בקשת GET

במהלך עשרים השנים האחרונות, "אפליקציות web" (כינוי לאפליקציות ושרתים שעושים שימוש בפרוטוקולי אינטרנט) צברו יותר ויותר פופלריות, ובמקביל התפתחו באופן מואץ היכולות, רמת התחכום והמורכבות של מערכות אלו. אתרי אינטרנט כבר לא הסתפקו בהגשה של דפי html ותמונות ערוכים מראש, אלא עשו שימוש בקוד ותכנות כדי לייצר דפי תוכן "דינמיים".

לצורך המחשה, חשבו מה קורה כשאתם מתחברים ל-Facebook - אתם רואים דף אינטרנט ובו דברים קבועים - סרגל כלים, לינקים - והרבה דברים שנקבעים מחדש בכל כניסה - למשל הודעות על פעילות של החברים שלכם ופרסומות. למעשה מאחורי הקלעים מסתתר שרת, שמקבל החלטות איך להרכיב את דף התוכן עבור כל בקשה של כל משתמש.
כך כאשר אתם פונים ל-Facebook, השרת צריך להחליט כיצד לבנות עבורכם את הדף. לאחר שהוא מבין מי המשתמש שפנה ומה אותו משתמש צריך לראות, הוא בונה עבורו את הדף ומגיש אותו ללקוח.

השירות ש-Facebook מספק מתוחכם בהרבה מעמודי אינטרנט בראשית הדרך, שבהם היה תוכן סטטי בלבד - קבוע מראש, והשרת רק היה מגיש את העמודים הללו ללקוחות.

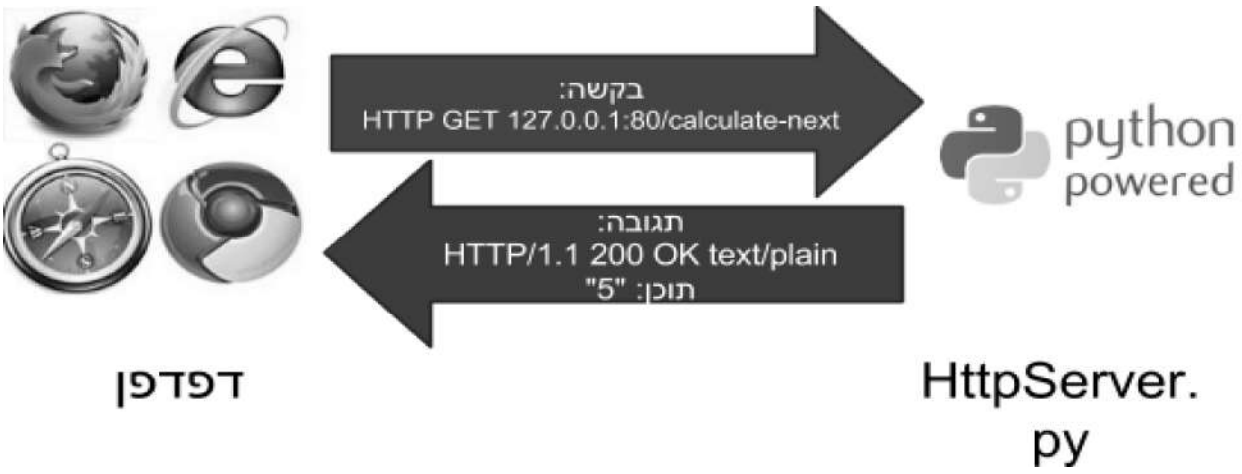
כעת נממש שרת שמגיב בתשובות תכנותיות לבקשות GET. כלומר, השרת שלנו יבנה תשובה באופן דינאמי בהתאם לבקשה שהגיעה מהלקוח. בינתיים, נממש את צד השרת, ולצורך צד הלקוח נמשיך להשתמש בדפדפן.

תרגיל 4.5

עדכנו את קוד השרת שכתבתם, ככה שפנייה לכתובת "/calculate-next" לא תחפש קובץ בשם הזה, אלא תענה על השאלה "מה המספר שמגיע אחרי 4". כלומר, השרת פשוט יחזיר תמיד "5" בתור תוכן התגובה לבקשה הזו.
הריצו את השרת ובדקו שהמספר 5 מתקבל כתגובה לפנייה לכתובת זו.

בתרגילים הבאים, שימו לב לכלול Headerים רלבנטיים בתשובותיכם, ולא "סתם" Headerים שאתם רואים בהסנפות. כדי שהתשובה תהיה תקינה, ציינו קוד תשובה (כגון 200 OK). עליכם לכלול גם Content-Length (שצריך להיות גודל התשובה, בבתים, ללא גודל ה-Headerים של HTTP). ציינו גם את ה-Content-Type (במקרה שתחזירו קובץ – סוג הקובץ שאתם מחזירים, שאפשר להסיק לפי סיומת הקובץ). תוכלו להוסיף גם Headerים נוספים, אך עשו זאת בצורה שתואמת את הבקשה והתשובה הספציפיות.

אם ביצעתם את התרגיל נכון, זה מה שבעצם קרה כאן:

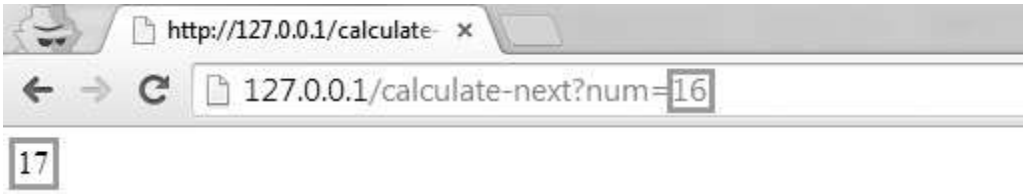


בטח כבר עליתם בעצמכם על כך שהשרת במקרה הזה הוא מאד לא שימושי, כי הוא מניח שהמשתמש תמיד מעוניין לדעת מה המספר שבא אחרי 4. בתכניות שכתבנו בפייתון, בדרך כלל עשינו שימוש בקלט מהמשתמש כדי לעשות איתו חישוב כלשהו במהלך התכנית. השרת שלנו היה הופך להרבה יותר שימושי, אם האפליקציה הייתה יכולה לבקש את המספר שבא אחרי מספר כלשהו; למעשה, אפשר לחשוב על השרת שלנו כמו על כל תכנית בפייתון שכתבנו עד היום, למעט העובדה שלא ניתן להעביר לתכנית קלט, ולכן היא מחזירה תמיד את אותו הפלט. אם נוכל להעביר לה בכל בקשה קלט שונה, נקבל את הפלט המתאים לכל קלט.

אם כך, כעת נעביר קלט לשרת באמצעות פרוטוקול ה-HTTP - למעשה נשתמש במה שנקרא " HTTP GET Parameters" - פרמטרים בבקשת ה-GET.
הדרך שבה מעבירים פרמטרים בבקשת GET היא באמצעות תוספת התו "?" בסיום הכתובת (זה התו שמשמש בפרוטוקול HTTP כמפריד בין הכתובת של המשאב לבין משתני הבקשה), ולאחר מכן את שם הפרמטר והערך שלו.

תרגיל 4.6

עדכנו את קוד השרת שכתבתם, כך שפנייה לכתובת "/calculate-next?num=4" תתפרש כפנייה לכתובת calculate-next עם השמה של הערך '4' במשתנה num, ולכן תחזיר את המספר 5.
אבל, "/calculate-next?num=16" תחזיר את התשובה '17',
"/calculate-next?num=10000" תחזיר את התשובה '10001',
והבקשה "/calculate-next?num=-200" תחזיר את התשובה '-199',

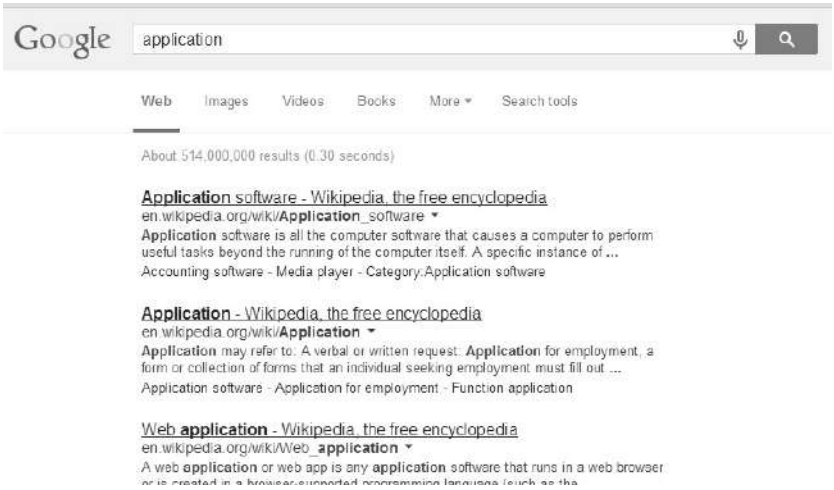


זוהי למעשה הדרך שבה יכולה אפליקציה להעביר פרמטרים כחלק מהבקשה שלה.

כדי להמחיש את העניין הזה, נתבונן בדוגמה מעניינת יותר, ולאחר מכן נחזור להבין את קטע הקוד של השרת שבו השתמשנו:

הכניסו את השורה הבאה בדפדפן: http://www.google.com/search

זוהי למעשה בקשת GET שנשלחת אל שרת החיפוש של Google. אבל, אתם עדיין לא רואים תוצאות חיפוש, כי ה"אפליקציה" (במקרה זה, האתר של Google בדפדפן שלכם), לא יודעת מה אתם רוצים לחפש. ברגע שתתחילו להכניס מילת חיפוש (נניח: "application"), נתחיל לראות תוצאות חיפוש במסך.



איך הגיעו התוצאות? הדפדפן שלח בקשת GET אל השרת של גוגל.

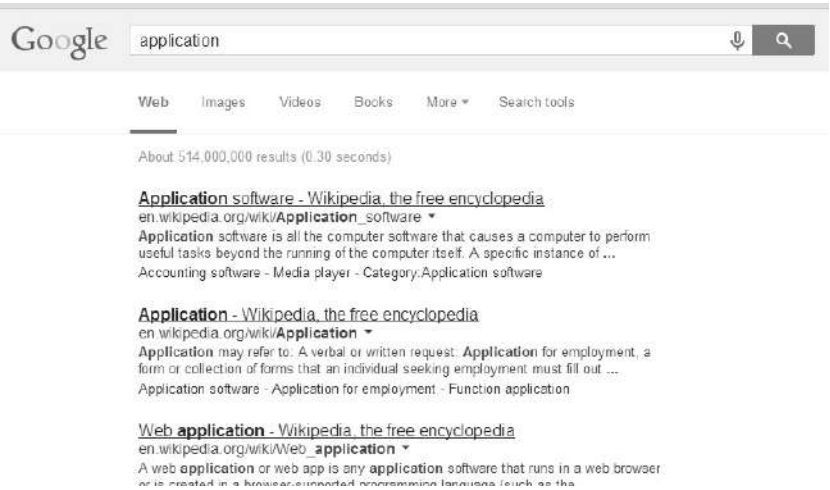
איך נראתה הבקשה? טוב ששאלתם, היא נראתה בדיוק כך[24]:

GET www.google.com/search?q=application HTTP/1.1

נסו בעצמכם להכניס את ה-URL בדפדפן (URL בלבד, ללא המילה GET וללא גירסת הפרוטוקול), ותראו בעצמכם מה קורה.

[24] ברוב המקרים ה-Host (בדוגמה זו "www.google.com") לא יכלל בשורת ה-GET, אלא ב-HTTP Header נפרד. עם זאת, על מנת שהדוגמאות תהיינה ברורות, אנו נציין אותו באופן מפורש.

לאחר שבדקתם בדפדפן - פיתחו **telnet** מול הכתובת www.google.com בפורט 80 (כמו שעשינו בתרגיל 4.3 של פרק זה), ובצעו את הבקשה הזו. נסו להבין את התגובה.



תרגיל 4.7

בכל אחת מהשורות בטבלה הבאה, תמצאו כתובת של משאב באינטרנט (אמנם עדיין לא הסברנו את המשמעות הפורמלית של המילה "משאב", אך ראיתם דוגמאות למשאבים, למשל www.google.com/search), שם של פרמטר, ורשימת ערכים.

מה שעליכם לעשות הוא להרכיב בקשת GET בכל אחד מהמקרים (כפי שראינו שבונים בקשה ממשאב, שם של פרמטר וערך שלו), לנסות אותה בדפדפן שלכם, ולכתוב בקצרה מה קיבלתם.

משאב	פרמטר	ערכים	בקשת GET	הסבר
google.com/maps	q	jerusalem, new-york, egypt		
twitter.com/search	q	madonna, obama		
wikipedia.org/w/index.php	search	israel, http, application		
youtube.com/results	search_query	obama, gangam, wolf		

נקודה למחשבה: שימו לב להבדל המרכזי בין ה-URLים הללו לבין URLים שעליהם התבססתם בתרגיל שרת http שמימשתם בסעיף הקודם - כיום "נעלמו" החלקים מה-URL שמתארים מיקום במערכת הקבצים, ואת מקומם החליפו הפרמטרים. זוהי תוצאה של השינוי שהתרחש באינטרנט בעשרים השנים האחרונות - מעבר מהגשה של דפי תוכן שנוצרו מראש, לבנייה של תשובות על סמך פרמטרים וקוד תוכנה שמרכיב את התשובה.

כעת נחזור לאחד החיפושים שעשינו ב-Twitter בתרגיל האחרון; שימו לב שהתוצאות שקיבלנו מכילות ציוצים (tweets) שהכילו את המילה obama. מה אם הייתי רוצה לחפש את הפרופיל של אובמה? (שימו לב שיש בצד שמאל כפתור שעושה את זה). האופן שבו ה"אפליקציה" (במקרה שלנו, האתר) של Twitter עושה את זה, הוא שהוא שולח פרמטר נוסף לשרת.

פרמטר אחד יכיל את מילת החיפוש (obama), והפרמטר השני יכיל את סוג החיפוש - חיפוש משתמשים (users). בין שני הפרמטרים יופיע המפריד &. נראה ביחד איך זה עובד.

הכניסו את השורה הבאה בדפדפן: https://twitter.com/search?q=obama&mode=users

באופן דומה, ניתן לחפש תמונות של אובמה, על ידי החלפת הערך של הפרמטר mode בערך photos - בנו את הבקשה ונסו זאת בעצמכם.

תרגיל 4.8

כמו שאתם וודאי מתארים לעצמכם, ניתן להעביר בבקשת ה-GET גם יותר משני פרמטרים, באותה הצורה. הרכיבו את בקשת ה-GET לשרת המפות של Google (בדומה לדוגמה שעשיתם בתרגיל הקודם), אך במקום חיפוש כתובת, בקשו את הוראות הנסיעה מתל אביב לירושלים בתחבורה ציבורית, על ידי שימוש בפרמטרים הבאים:

- saddr - כתובת המוצא (למשל: 'Tel+Aviv').
- daddr - כתובת היעד.
- dirflg - אמצעי התחבורה. תוכלו להעביר 'r' עבור תחבורה ציבורית (או 'w' אם אתם מתכוונים ללכת ברגל).

בנו את בקשת ה-GET, הכניסו אותה בדפדפן שלכם, וודאו שאתם מקבלים את הוראות הנסיעה.

תרגיל 4.9

כתבו שרת HTTP משלכם, שמחשב את השטח של משולש, על סמך שני פרמטרים: גובה המשולש והרוחב שלו. למשל, עבור שורת הבקשה הבאה: http://127.0.0.1:80/calculate-area?height=3&width=4, יחזיר "6.0". בדקו אותו גם עבור קלטים נוספים.

פרוטוקול HTTP - בקשות POST ותכנות צד לקוח בפייתון (הרחבה)

נפתח בשאלה - האם העליתם פעם תמונה ל-Facebook? קל להניח שכן (ואם תתעקשו שמעולם לא עשיתם זאת, ניתן להניח במקום כי שלחתם קובץ PDF או מסמך Word למישהו במייל, או לפחות מילאתם טופס "יצירת קשר" באתר כלשהו). המשותף לכל המקרים שתיארנו הוא - הצורך להעביר כמות מידע **מ**האפליקציה **אל** השרת, כמות של מידע שלא ניתן להעביר בבקשת GET.

נתעכב כדי להבין טוב יותר את ההבדל בין בקשת GET עם פרמטרים לשרת של Twitter, שמחזירה דף אינטרנט עם תמונות של אובמה (כמו שראינו בסעיף הקודם), לבין בקשה לשרת של Facebook, שתאחסן שם תמונה.

במקרה שכבר ראינו - שליחת בקשה לתמונות באתר Twitter - האפליקציה (הדפדפן שמציג את עמוד האינטרנט של Twitter) רוצה להציג למשתמש את התמונות שמתאימות לחיפוש "אובמה":

.

כדי לעשות זאת, נשלחת לשרת בקשה די קצרה, שמכילה בסך הכל את הכתובת "twitter.com/search", ושני פרמטרים – obama, photos. כל זה נכנס בפקטה אחת. התשובה, לעומת זאת, מכילה הרבה מידע, ואם נסניף את התקשורת, נראה שהתשובה מורכבת ממספר גדול של פקטות. עשו זאת, ומצאו את פקטת הבקשה, וכמה פקטות היה צריך כדי להעביר את כל התשובה.

מה ההבדל לעומת המקרה שבו נעלה תמונה ל-Facebook?

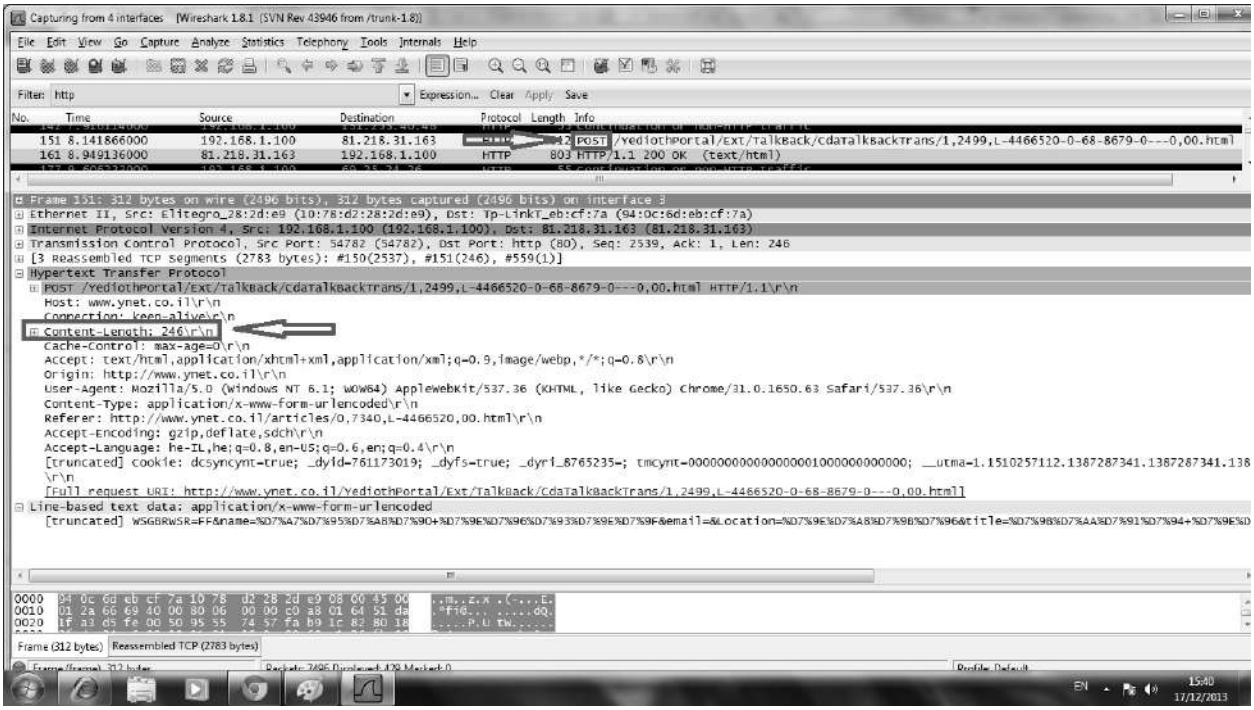
במקרה זה, המידע נמצא בצד האפליקציה (ספציפית - התמונה. בטלפון ממוצע תמונה יכולה להיות בגודל של יותר מ-1MB), והיא מעוניינת להעביר את כולו לשרת.

האם ניתן להשתמש בפרמטרים של בקשת GET כדי להעביר את כל התמונה?

אם ניקח בחשבון את העובדה שהאורך המקסימלי של URL בו תומכים רוב הדפדפנים הוא 2,000 תווים, וה-URL הוא כל מה שניתן להעביר בבקשת GET, די ברור שלא נוכל להעביר את התמונה לשרת באמצעות בקשת GET.

לכן, נעשה שימוש בסוג חדש של בקשה שקיים בפרוטוקול HTTP - בקשת POST.

כדי להתנסות בשימוש בבקשת POST, ניכנס לאתר ynet.co.il, נבחר את כתבה שמצאנו בה עניין, ונוסיף לה בתגובה - "כתבה מעניינת, תודה רבה." - רגע לפני שנלחץ על הכפתור "שלח תגובה", נפעיל הסנפה ב-Wireshark, ולאחר שנשלח את התגובה, נוכל למצוא פקטת HTTP עם בקשה מסוג POST, שנשלחה לשרת של Ynet:



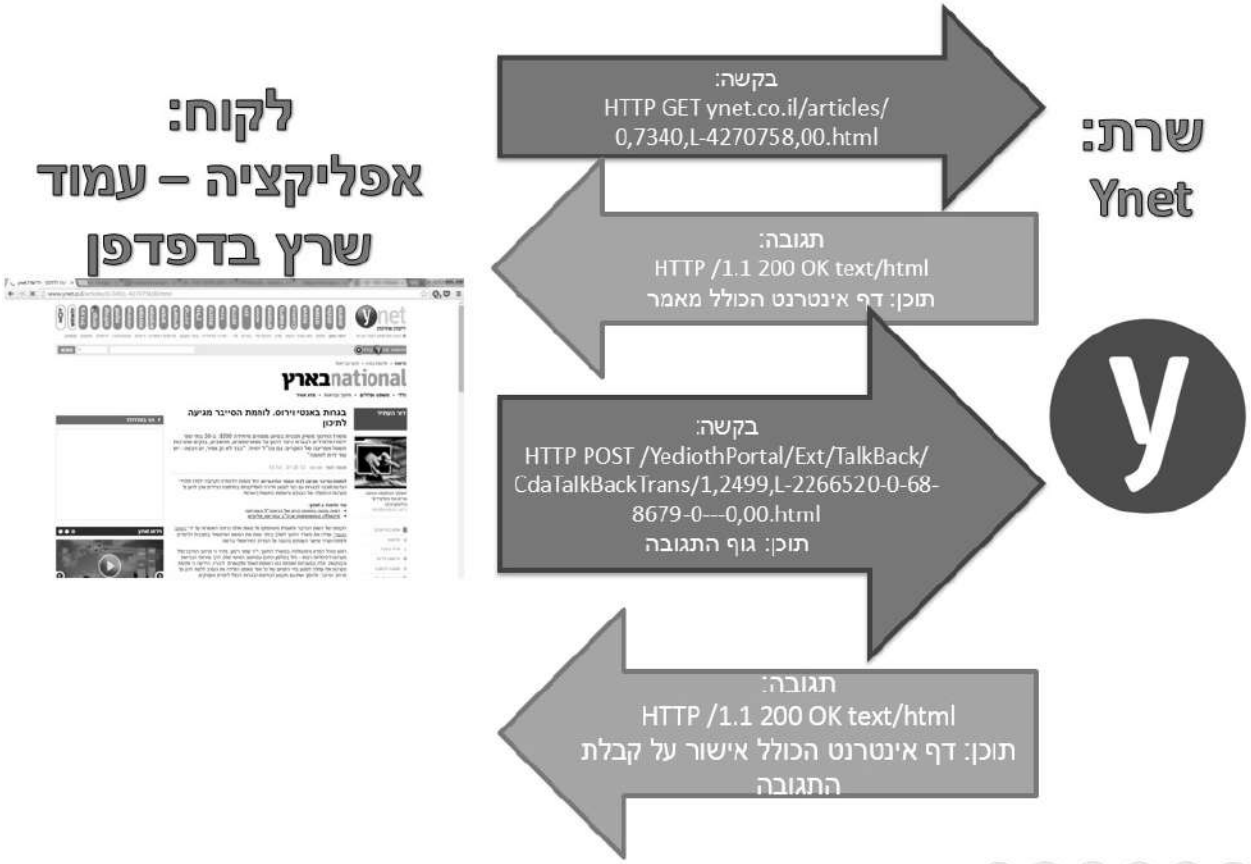
נוודא שאנחנו מבינים מה קרה כאן - בתגובה ששלחנו, מילאנו את השם, את מקום המגורים, ואת כתובת המייל שלנו. בנוסף, מילאנו כותרת לתגובה, ואת תוכן התגובה עצמו (שיכול להיות גם ארוך הרבה יותר מאשר "כתבה נחמדה"). האם היה ניתן לשלוח את כל הנתונים האלה לשרת באמצעות בקשה GET?

ynet.co.il/articles/17773?name=Moshe&address=Rehovot&mail=moshe&rehovot.co.il&title=nice_article&body=thank_you_this_was_a_great_article_...

זוהי אינה אופציה מציאותית - האורך המקסימלי של URL הוא 2,000 תווים, בעוד שרק התגובה עצמה עשויה להיות יותר ארוכה מכך...

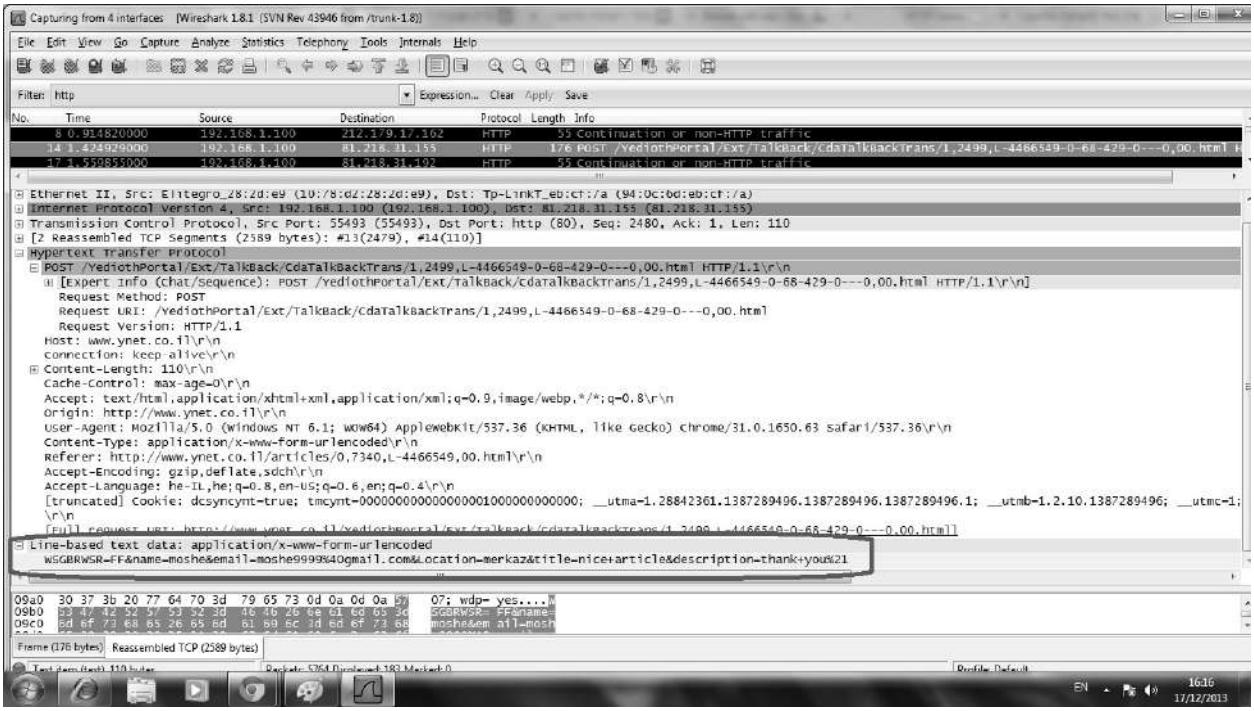
לכן, בנוסף לפרמטרים שעוברים כחלק מהכתובת (URL), בבקשת POST ניתן לצרף **תוכן**, ושם יועבר תוכן התגובה.

נביט בתהליך המלא שבוצע מול השרת של Ynet:



באופן כללי, אחד השימושים הנפוצים בבקשות POST הוא בטפסים - הכוונה היא לטופס שמכיל מספר שדות, שאותם המשתמש ממלא ואז לוחץ על כפתור כדי לשלוח את הטופס - זהו בדיוק המקרה עם תגובות באתר Ynet. דוגמה נוספת - אם תלחצו על "צור קשר" באתר של חברת הסלולרי שלכם, ותמלאו שם את הטופס, התוכן יישלח לשרת של אתר האינטרנט שלהם באופן דומה.

אם תרצו "לראות" היכן עובר התוכן עצמו בבקשת ה-POST - לטפסים יש דרך די סטנדרטית להעביר את הנתונים האלה. הדרך הזו קצת מזכירה את האופן שבו משתמשים בפרמטרים ב-URL, רק שבמקרה זה, אין מגבלה של מקום. כדי לראות זאת - ב-Wireshark התבוננו בתוכן של פקטת ה-POST, וחפשו את text-data. נסו למצוא שם את השדות השונים שמילאתם בתגובה (שמכם, מקום המגורים, הכותרת וכו').



תרגיל 4.10

מכיוון שבקשות POST לא נוכל לייצר באמצעות שורת הפקודה בדפדפן, נצטרך לכתוב תכנית בפייתון שתדמה גם את צד הלקוח בתקשורת - כלומר את האפליקציה.

כתבו תוכנית חדשה שתתפקד בתור לקוח HTTP. התוכנית תשלח בקשת POST אל השרת (תוכלו לבחור את הכתובת בעצמכם, למשל /upload). ה-Body של בקשת ה-POST יכיל את התוכן של התמונה שיש לשמור בתיקיית התמונות. את שם הקובץ לשמירה ציינו בתור פרמטר בשם "file-name".
בנוסף, ממשו בשרת את קבלת בקשת ה-POST ושמירת התמונה בתיקיית התמונות לפי שם הקובץ שהגיע.

בטח תטענו (ובצדק!) שאין שום טעם בשרת שרק ניתן לשמור אליו תמונות, אם לא ניתן לבקש אותן בחזרה.

תרגיל 4.11

הוסיפו לשרת שכתבתם תמיכה בבקשת GET תחת המשאב image, שמקבלת פרמטר בשם image-name, ומחזירה את התמונה שנשמרה בשם הזה (או קוד תגובה 404, במידה שלא קיימת תמונה בשם הזה).

הריצו את השרת החדש, ובצעו פקודת POST (אחת או יותר) כדי להעלות תמונות לשרת. קראו לאחת התמונות בשם "test-image".

כעת, פתחו את הדפדפן (בו בדרך כלל השתמשנו עד כה בתור צד לקוח ליצירת בקשות GET), והכניסו את השורה הבאה: http://127.0.0.1:80/image?image-name=test-image

אחרי שראיתם את התמונה שהעליתם מוקדם יותר חוזרת מהשרת, וודאו שהבנתם עד הסוף מה בעצם קרה כאן.

HTTP - סיכום קצר

עד כה, למדנו כיצד להתנהל עם משאבים באמצעות פרוטוקול HTTP. בתחילה, למדנו לבקש משאבים על סמך השם שלהם באמצעות GET; לאחר מכן למדנו לצרף פרמטרים נוספים כדי "לחדד" את הבקשה שאיתה אנו פונים אל משאב הרשת - לדוגמה, כשפנינו אל שירות המפות עם חיפוש המסלול מת"א לירושלים, הוספנו פרמטר נוסף לבקשה שמסמן שהמסלול צריך להיות בתחבורה ציבורית.
הבנו כי בקשה של משאב לא אמורה לשנות דבר בצד השרת - רק להחזיר תוצאה מסויימת. ניתן לחזור על אותה הבקשה מספר רב של פעמים, והתוצאה אמורה להיות זהה.

לעומת זאת, בהמשך למדנו להעלות של מידע מצד האפליקציה אל השרת באמצעות POST. פעולה זו בהחלט גורמת לשינוי במידע שנמצא בשרת, כפי שראינו בשרת אחזור התמונות שכתבנו בסעיף הקודם. אני יכול לבקש תמונה בשם "december-21" ולקבל הודעת שגיאה שאומרת שהמשאב אינו נמצא (404), וביום שלאחר מכן לבצע שוב את אותה הבקשה, אלא שהפעם אקבל תמונה בחזרה.
מה ההסבר לכך? כנראה שבינתיים מישהו ביצע פקודת POST והעלה תמונה בשם הזה.

הבנו כי מאחורי משאבי האינטרנט נמצאים שרתים שמריצים קוד - בדומה לשרת שכתבנו בפרק זה - ומשתמשים בו כדי לייצר תגובות לבקשות GET ו-POST. אבל, שרתים כאלה הם לרוב מורכבים הרבה יותר מהשרת שכתבנו, ולרוב ישתמשו גם בבסיס נתונים (database) - יכתבו אליו בבקשות POST, ויקראו ממנו בבקשות GET. הם גם יידעו איך לתמוך במספר משתמשים (לקוחות) שמבקשים בקשות בו זמנית, יפעילו מנגנוני אבטחה והרשאות, ויתמכו בסוגי בקשות נוספים (כמו למשל בקשות למחיקה).