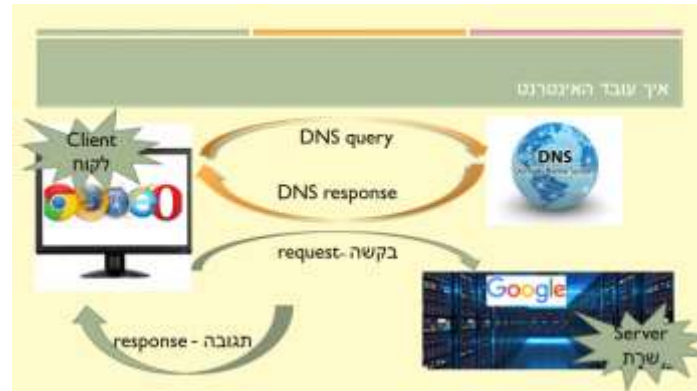




HTTP Server

1 התרגיל

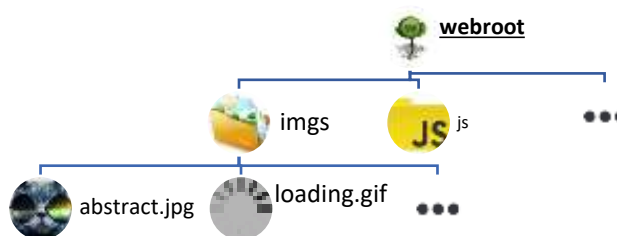
בתרגיל שלפנינו נקים אתר interne כמו Facebook או Ynet



אנו נכתוב את השרת בעוד שהלקוח שלנו יהיה הדפדפן. כדי לבדוק את התרגיל נפתח את הדפדפן ונכתוב בחלונית לדוגמא:



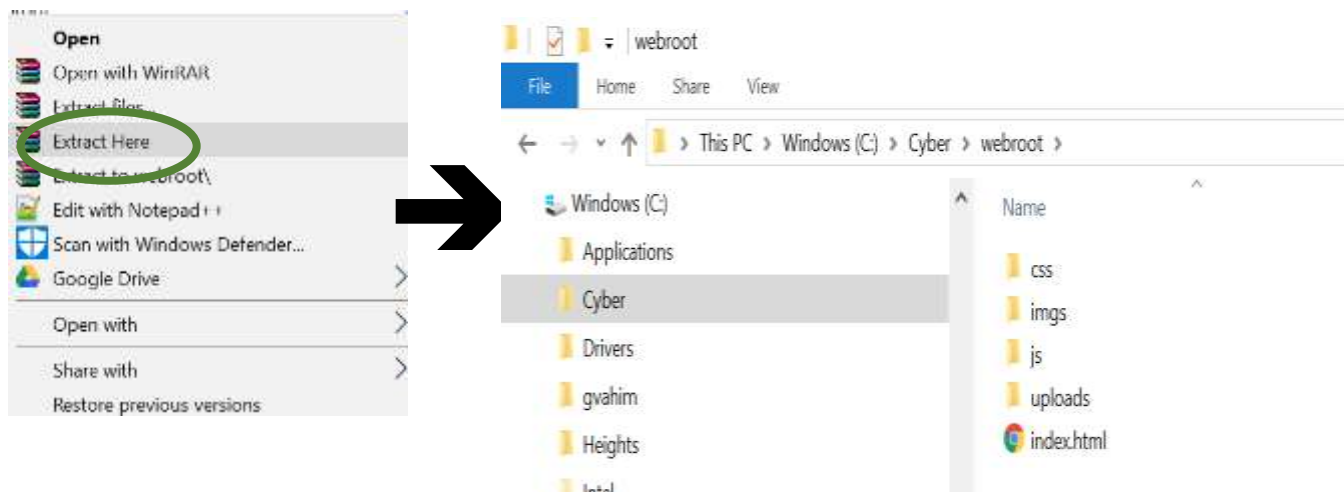
אם מתחת ל webroot שלנו, יש תיקיית imgs שבה קובץ abstract.jpg שמכיל את תמונת החתול שלפנינו, כתשובה נצפה לראות את המשאב אותו ביקשנו. במקרה של imgs/abstract.jpg נצפה לראות בדפדפן:



2 הקמת האתר הפיזי

לפני שניגש לכתיבת הקוד נקים את האתר הפיזי. בכל אתר מוחזקים משאבים (קבצים) שאותם האתר מספק מתחת לתיקיית webroot. נקים תיקיית webroot משלנו ובתיקות שמתחת נשים את המשאבים.

הכנתי עבורכם תיקייה ובה משאבים. הורידו אל המחשבים שלכם את הקובץ webroot.zip מהמודל. שימו אותו בתיקייה ידועה על ה diskonkey או המחשב הנייד שלכם ופתחו אותו. לחצן ימני על העכבר ← ובחרו Extract Here. תיווצר תיקיית webroot ומתחתיה תיקיות נוספות המכילות קבצים – אלו המשאבים שלנו



3 כתיבת ה server

נכתוב כעת את השרת שפותח socket ומקבל בקשות למשאבים שב webroot.

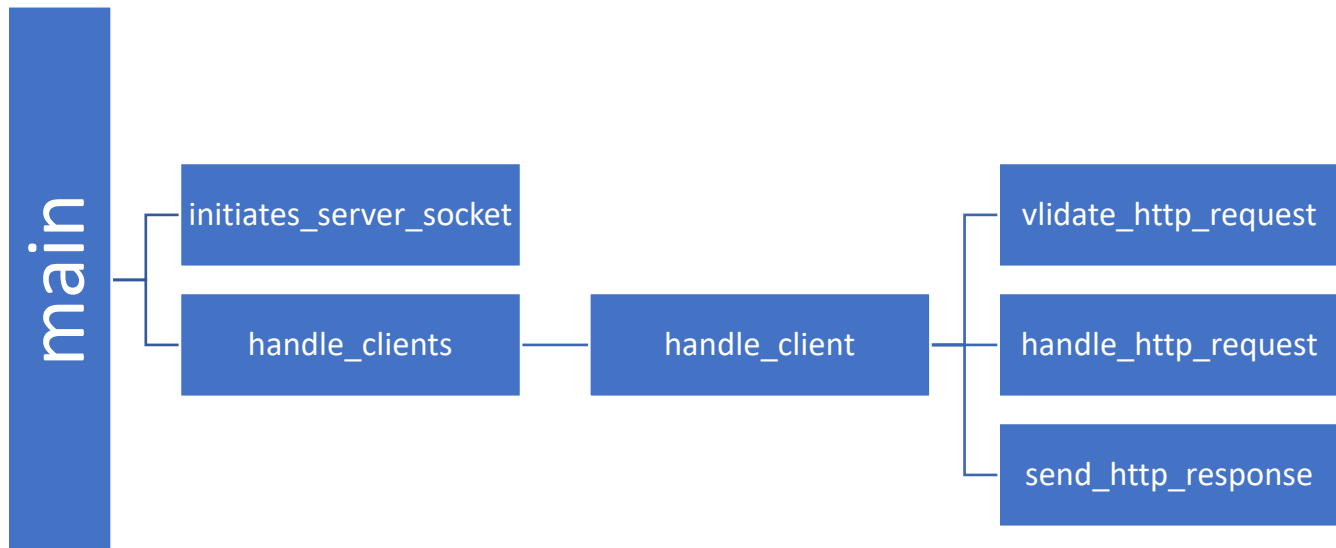
נחלק את הכתיבה ל 6 שלבים:

- i. הקמת ה server
- ii. ה server שלנו יחזיר את תוכן הקבצים
- iii. החזרת html index
- iv. ה server שלנו יטפל בשגיאות
- v. ה server שלנו יטפל בהזזה של קובץ - redirect
- vi. ה server שלנו יחזיר גם תוצאות של פעולות חישוביות

♥ שימו לב! בסוף כל שלב עליכם לבדוק את ה Server שלכם בעזרת הדפדפן

3.1 הקמת ה Server

בשרת שלנו כמו בכל שרת אחר נפתח socket ונאזין עליו. עבור כל לקוח שיתחבר נטפל בבקשות שיתקבלו ונחזיר תשובה. לכן הבסיס שלו מאד דומה לבסיס של השרת טכנאי. בצורה סכמתית מבנה השרת שלנו הוא כזה:



3.1.1 main

נתחיל ב main שנראית בדיוק כמו ה main בשרת טכנאי

```

def main():
    """
    server main - receives a message returns it to
    client
    """
    try:
        server_socket = initiate_server_socket()
        handle_clients(server_socket)
        server_socket.close()
    except socket.error as msg:
        print("socket failure: ", msg)
    except Exception as msg:
        print("exception: ", msg)
  
```

בדיוק כמו בשרת הרגיל ב main נאתחל את ה socket ע"י זימון הפעולה initiate_socket ונקרא לפעולה handle_clients שתחכה בלולאה להחברות לקוחות.

ולא נשכח כמובן טיפול ב exceptions. הכניסו את תוכן הלולאה שב main ל try ו except של socket.error כך שאם יהיה ניתוק או timeout ה exception יתפס ונחזור לעשות accept. סיגרו את ה server_socket ב except.

שימו לב: אנו משתמשים ב sockets לכן יש להוסיף בראש הקובץ import socket



80

3.1.2 הקמת ה CONNECTION initiate_server_socket

- נקבע את ה ip וה port של ה server. ה port של שרת האינטרנט שלנו כזכור לנו הוא תמיד 80 היות ואתם רוצים לקבל כל מי שיפנה למחשב שלכם נקבע את ה IP להיות מחשב – '0.0.0.0'

IP = "0.0.0.0"

PORT = 80

שימו לב, IP ו PORT שניהם קבועים ולכן יש להגדירם בראש הקובץ באותיות גדולות. כתבו בראש הקובץ:

```
def initiate_server_socket():
    server_socket =
    socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    server_socket.bind((IP, PORT))
    server_socket.listen(10)
    print("Listening for connections on port %d" % PORT)
    return server_socket
```

נאתחל את ה socket כמו בשרת טכנאי. מה יהיה ערך ה port שלנו? (http port = ?)

נאפשר הפעם ל 10 לקוחות לחכות בתור

3.1.3 טיפול בלקוחות שהתחברו – handle_clients

- נוסיף SOCKET_TIMEOUT. בדקו באינטרנט אילו יחידות זמן הפעולה מקבלת (דקות, שניות, מילישניות) והגדירו את ה timeout לפרק זמן קצר כתבו בראש הקובץ:

SOCKET_TIMEOUT = 1

```
def handle_clients(server_socket):
    """ loop forever while waiting for clients """
    client_socket = None
    while True:
        try:
            client_socket, client_address = server_socket.accept()
            print('New connection received')
            client_socket.settimeout(0.5)
            handle_client(client_socket)
        except socket.error as e:
            print("client failed", e)
            client_socket.close()
```

לולאה שמחכה ל clients כש client מתחבר, נקראת הפעולה handle_client שעונה לבקשות ה client עד לניתוק ה connection

שימו לב ל timeout. לפעמים הקשר בין השרת ללקוח נתקע וה timeout משחרר אותו. שימו לב שהוא גורם לexception שבו יש לטפל

3.1.4 קבלת בקשות מהלקוח – hanle_client

נשאר לנו לטפל בקבלת בקשות מהלקוח. הוסיפו לפעולה handle_client את קבלת ההודעות מהלקוח.

הוסיפו בראש הקובץ את הקבוע

MSG_SIZE = 1024

```
def handle_client(client_socket):
    print('Client connected')
    done = False
    while not done:
        raw_client_request = client_socket.recv(MESSAGE_SIZE)
        client_request = raw_client_request.decode()
        done = (client_request == "")
        if not done:
            valid_http, resource = validate_http_request(client_request)
            if valid_http:
                handle_client_request(resource, client_socket)
            else:
                print('Error: invalid HTTP request')
                done = True
        else:
            print('connection closed')

    print('Closing connection')
    client_socket.close()
```

טיפול בבקשות client עד שמגיעה בקשה שאינה תקינה שימו לב שעליכם לקבל את ההודעות מהלקוח אם התנתק ה connection נקבל מהלקוח " עבור כל הודעה הפעולה קוראת ל:

- validate_http_request
- handle_client_request

בשלב הראשון אם נקבל הודעה לא חוקית ננתק את הקשר – כלומר נצא מהלולאה: (done = True) ונסגור את ה socket

כעת הבקשה נמצאת ב client_request. עלינו לוודא שהבקשה תקינה ולהחזיר את המשאב המבוקש

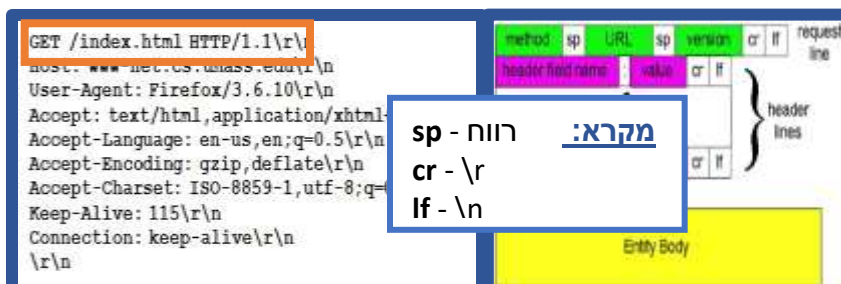
3.2 טיפול בבקשה

3.2.1 בדיקת חוקיות - validate_http_request

לאחר שקיבלנו בקשה מהלקוח עלינו לטפל בה. נחלק את הטיפול ל 2 שלבים

בדיקת חוקיות – ממשו את הפעולה validate_http_request כך שתבדוק את חוקיות הבקשה ותחזיר tuple. אם הבקשה חוקית הפעולה תחזיר (שם המשאב, True) ואם אינה חוקית תחזיר (False, None)

ניזכר במבנה הבקשה:



הבקשה שלנו חוקית אם:

- המילה הראשונה בה GET
- המילה השנייה בה היא נתיב למשאב (קובץ) שנמצא מתחת ל webroot
- המילה השלישית בה היא HTTP/1.1

שימרו את מיקום התיקייה webroot בקבוע. בדקו בפעולה validate_http_request אם הבקשה עונה על שלשת התנאים

בדקו גם אם המשאב המבוקש הוא אכן קובץ שנמצא במיקום המבוקש ב webroot

שם הקובץ: נגזור את שם המשאב מה URL וכדי לקבל את מיקום הקובץ נחבר אותו עם ה root (webroot) לדוגמא אם התיקייה webroot נמצאת ב c:\cyber וקיבלנו בקשה ל: 127.0.0.1/imgs/abstract.jpg הרי שהקובץ שלנו הוא: c:\cyber\webroot\imgs\abstract.jpg

```
def validate_http_request(request):
```

```
    """
```

```
    validates the http request:
```

```
    1- starts with GET
```

```
    2 - ends with HTTP/1.1
```

```
    3 - holds a valid resource name between both of them
```

```
    returns a tuple -
```

```
    boolean: isValid
```

```
    retrieved resource name
```

```
    """
```

```
    # your code here
```

בפונקציה עליכם לבדוק את החוקיות של בקשת HTTP

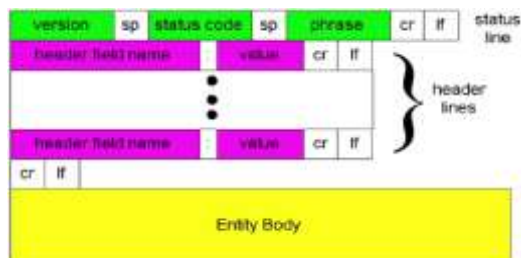
נחזיר מהפרוצדורה 2 ערכים:

- האם הבקשה חוקית
- את שם ה resource

3.2.2 בניית והחזרת תגובה handle_client_request

ניזכר במבנה התגובה (response)

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```



כלומר, התגובה שלנו תכיל:

- HTTP/1.1 200 ok\r\n
- Content-Length: <data length> \r\n
- Content-Type: <contrnt type> \r\n
- \r\n
- Data

שורת התגובה

אורך הנתונים המוחזרים

סוג הנתונים המוחזרים – ייקבע לפי שם הקובץ

הנתונים המוחזרים - תוכן הקובץ

קובץ	Content-Type
File.html, File.txt	text/html; charset=utf-8
File.jpg, File.ico, File.gif, File.png	image/jpeg
File.css	text/css
File.js	text/javascript; charset=utf-8

Content-Type – נקבע על פי סיומת הקובץ
על פי החוקים שבטבלה:

כלומר ההודעה שלנו תהיה:

HTTP/1.1 200 ok\r\n Content-Length: <אורך תוכן הקובץ>\r\n
Content-Type: <תוכן הקובץ>\r\n<מחושב על פי הטבלה>

בנו את ההודעה באופן הבא:

VERSION = "HTTP/1.1"
EOL = "\r\n"

```
data = file_content.read in get_file_data
status = "200 ok"
content_len = len(data)
content_type = calculated by the above table rules
http_header = VERSION + " " + status + EOL +
              "Content-Length: " + str(content_len) + EOL +
              "Content-Type: " + content_type + EOL +
              EOL
http_response = http_header.encode() + data
```

שימו לב ש data אותה קראתם מהקובץ היא מטיפוס byte array ולכן לא ניתן לחבר אותה למחרוזת
http_header

```
def handle_client_request(resource, client_socket):
    print(' requested resource: ' + resource)
    # retrieve file content and content type
    data, content_type_str = file_request(resource)
    data_size = str(len(data))

    # finally constructing header
    http_header = 'HTTP/1.1 ' + "200 OK " + EOL + \
                  'Content-Length: ' + data_size + EOL + \
                  content_type_str + \
                  EOL + EOL

    print(http_header)
    http_response = http_header.encode() + data
    client_socket.send(http_response)
```

בפונקציה עליכם לזמן פעולות
שתכתב על מנת לקבל את תוכן
המשאב ואת ה content
type ובאמצעותם לבנות את
התגובה לבקשה ולשלוח אותה

3.3 החזרת index.html

הקובץ index.html הוא קובץ מיוחד. אם השרת מקבל בקשת GET ל root - (כלומר למיקום "/") - נחזיר את index.html

הגדירו את הקבוע DEFAULT_RESOURCE לדוגמא:

DEFAULT_RESOURCE = 'C:\\Cyber\\webroot\\index.html'

```
if resource == '/'
    resource = DEFAULT_RESOURCE
```

בפעולה validate_http_request הוסיפו בדיקה ל resource כך שאם קיבלתם "/" תהפכו את ה resource kvhu, v ל DEFAULT_RESOURCE

3.4 טיפול בשגיאות

- החזירו בהודעת התגובה, ב status, 404 Not Found (במקום 200 OK) אם הקובץ לא קיים.
- החזירו 500 Server Internal Error, במקום לנתק את ה connection – במקרה והבקשה נראית לכם לא חוקית 5 נקודות
- הדרכה:
 - שנו את הפעולה validate_http_request כך שתחזיר מחרוזת של status במקום ערכים בוליאניים של valid | not valid:

- את ה status. אם היא חוקית ה status יהיה 200 OK
- אם המשאב לא קיים 404 Not Found
- אם קיבלתם הודעה לא חוקית אחרת החזירו 500 Server Internal Error

- שנו את הפעולה handle_client_request כך שתקבל את ה status. אם ה status שקיבלתם הוא 200 OK בנו את ההודעה כמו קודם. במידה ולא בנו את הודעת השגיאה לדוגמא: HTTP/1.1 404 Not Found\r\n\r\n

כלומר מבנה הודעת השגיאה:

http_response = VERSION + " " + status + EOL + EOL

- שנו את הפעולה handle_client_request שבמקום לנתק את ה connection (לצאת מהלולאה המטפלת בלקוח ולסגור את ה socket) במקרה שההודעה לא היתה תקינה תשלח לפעולה handle_client_request את מחרוזת ה status.

3.5 Redirect

פעולת ה redirect היא פעולת הפניה מחדש שמבצע ה server. אם מגיעה בקשה למשאב ש"הוזז" כלומר שנמצא ברשימת ה"redirect" ה server מחזיר 302 Moved Temporarily בשורת התגובה וב header נוסף השדה location שערכו המיקום הנכון של הקובץ מתחת ל root. client לאחר קבלת תשובה כזאת, מבקש מה server את המשאב במיקום הנכון.

שימו לב: השרת לא מחזיר את תוכן המשאב הנכון כי אם את המיקום שלו ועל הלקוח מוטל לבקש את המשאב מחדש במיקום הנכון

הלקוח מבקש משאב שהוזז GET <moved resource> HTTP 1.1 /r/n	מיקום חדש השרת מחזיר HTTP 1.1 302 Moved Temporarily\r\n location: <current location>\r\n\r\n	הלקוח מבקש את המשאב הנכון Get <current location> HTTP 1.1/r/n	השרת מחזיר את תוכן הקובץ HTTP 1.1 200 OK
--	--	--	--

הוסיפו REDIRECTION_DICTIONARY שמכיל את שמות המשאבים ש"הוזזו" ואת מיקומם החדש

לדוגמא:

```
REDIRECTION_DICTIONARY = {'/js/box1.js': '/js/box.js'}
```

במילון שלנו כתוב שהקובץ box1.js עבר ל box.js.

כעת כשתגיע בקשה לקובץ נבדוק האם הוא "הוזז" כלומר האם הוא נמצא ב dictionary ונחזיר header מתאים

- שנו את הפעולה validate_http_request כך שתחזיר 302 Moved Temporarily אם המשאב נמצא ב dictionary. הפעם החזירו את המשאב שכן עלינו עדיין לשלוח את המיקום הנכון ללקוח מ handle_client_request
- בפעולה handle_client_request בדקו אם ה status הוא 302 Moved Temporarily ואם כן שילפו אותו מה dictionary החזירו את ה header

```
file location = REDIRECTION_DICTIONAR[resource]
status = "302 Moved Temporarily"
http_response = VERSION + " " + status + EOL +
  "Location: " + file location + EOL+ EOL
```

3.6 פעולות חישוביות

<p>פעולות חישוביות ומשתנים</p> <ul style="list-style-type: none"> ניתן להכניס פרמטרים לכתובת ה-url העברת פרמטר לשרת מתבצעת ע"י הוספת התו '?' מספרי בין כתובת המשאב לפרמטרים של הבקשה נסו את ה-url הבא: <p>https://www.google.com/search?q=Israel</p>	<p>פעולות חישוביות ומשתנים</p> <ul style="list-style-type: none"> עד עכשיו קיבלנו שמות של קבצים והחזרנו את תוכנם. אנחנו יודעים שהאינטרנט עושה דברים הרבה יותר מסובכים לדוגמא Google מוצא תכנים המתאימים למילה שביקשנו איך זה עובד? <p></p>
	<p>פעולות חישוביות ומשתנים</p> <ul style="list-style-type: none"> ניתן להעביר דרך ה-url מספר פרמטרים, לפי הצורך נחפש בטוויטר את החשבון של הנשיא ראובן ריבלין <p>https://twitter.com/search?f=users&q=rvlin</p> <ul style="list-style-type: none"> כעת, אם נרצה לקבל תמונות של הנשיא ריבלין נחליף את המילה users במילה photos <p></p>

נגדיר בשרת שלנו 2 פעולות

- **calculate-next?num=x** – כשנקבל פנייה למשאב calculate-next נחזיר את x+1 לדוגמא: עבור <http://127.0.0.1/calculate-next?num=11> יוחזר 12
- ♥ נשים לב שה **content-type** במקרה הזה הוא **text/plain**
- **calculate-area?height=h&width=w** – כשנקבל פנייה למשאב calculate-area נחזיר את חישוב השטח של משולש $(w * h) / 2$, על סמך שני פרמטרים: גובה המשולש והרוחב שלו. לדוגמא: עבור <http://127.0.0.1/calculate-area?height=3&width=2> יוחזר $3*2/2$ כלומר "3.0"

הדרכה:

כתבו פעולות המתאימות לבקשות החישוביות


שנו את `validat_http_requeset` כך שבקשות שמכילות פנייה חוקית לפעולה יזוהו כחוקיות ואילו בקשות שאינן חוקיות יחזירו שגיאה

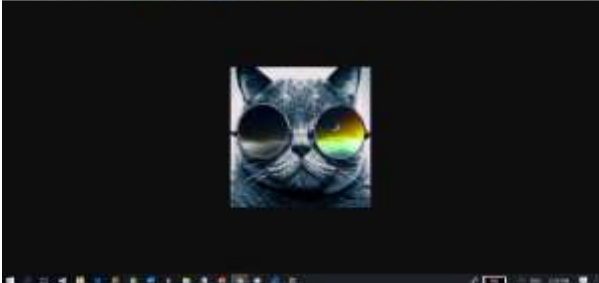

בדקו ב `handle_client_request` אם הגיעה בקשת לפעולה ם כן שלפו את הפרמטרים מהבקשה וזמנו את הפעולה עם הפרמטרים המתאימים

מזכירה לכם שעל מנת לעבוד על קטעי מחזות שונים ניתן להשתמש ב `split` או ב `find`

3.7 בדיקות

לפני ההגשה, אחרי שאירגנתם את הקוד (קבועים, תיעוד, pep8) בדקו בבקשה את הבדיקות הבאות:

תוצאה	url
	127.0.0.1
	בדקו בעזרת האתר את calculate calculate area next
	שילחו תו במקום מספר פעולות החשובות שימו לב שאתם מקבלים server internal error
	127.0.0.1/js/box1.js

	<p>הוסיפו למילון /imgs/abstract1.jpg: /imgs/abstract.jpg והריצו: /imgs/abstract1.jpg</p>
	<p>127.0.0.1/tralala</p>