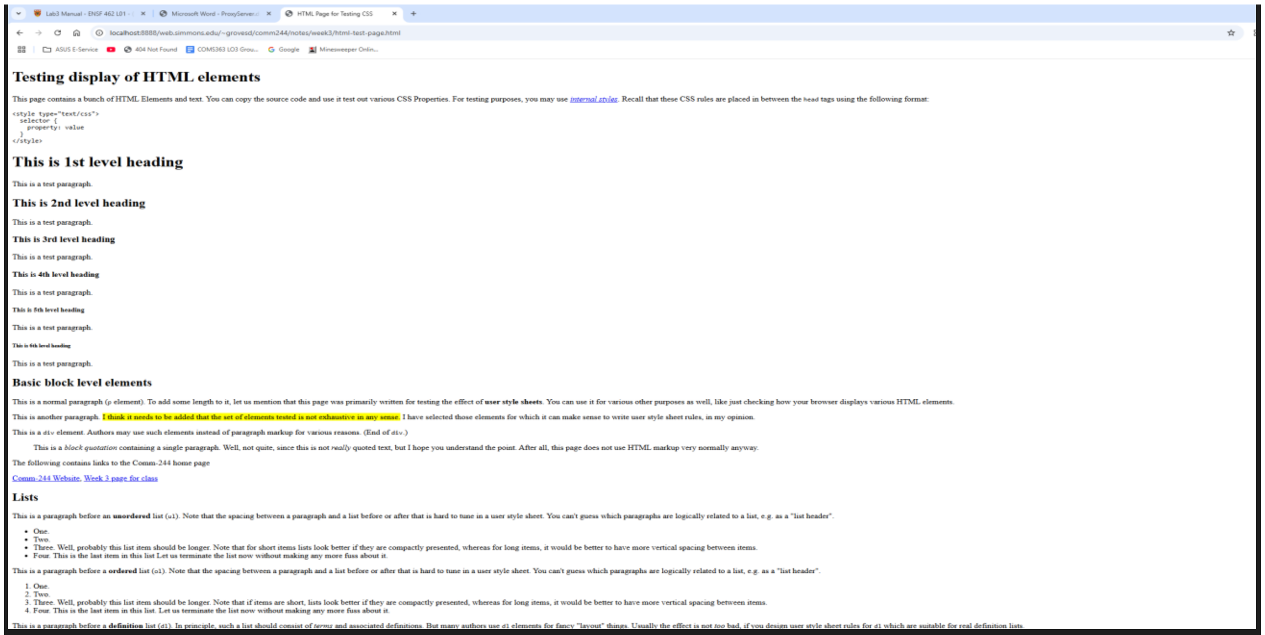
Name: Phoenix Bouma

UCID: 30139717


Lab3 Manual - ENSF 462 L01
Microsoft Word - ProxyServer.d
HTML Page for Testing CSS
localhost:8888/web.simmons.edu/~grovesd/comm244/notes/week3/html-test-page.html
ASUS E-Service
404 Not Found
COMS363 LO3 Grou...
Google
Minesweeper Onlin...

Testing display of HTML elements

This page contains a bunch of HTML Elements and text. You can copy the source code and use it test out various CSS Properties. For testing purposes, you may use *internal styles*. Recall that these CSS rules are placed in between the head tags using the following format:

```
<style type="text/css">
  selector {
    property: value
  }
</style>
```

This is 1st level heading

This is a test paragraph.

This is 2nd level heading

This is a test paragraph.

This is 3rd level heading

This is a test paragraph.

This is 4th level heading

This is a test paragraph.

This is 5th level heading

This is a test paragraph.

This is 6th level heading

This is a test paragraph.

Basic block level elements

This is a normal paragraph (p element). To add some length to it, let us mention that this page was primarily written for testing the effect of **user style sheets**. You can use it for various other purposes as well, like just checking how your browser displays various HTML elements.

This is another paragraph. I think it needs to be added that the set of elements tested is not exhaustive in any sense. I have selected those elements for which it can make sense to write user style sheet rules, in my opinion.

This is a div element. Authors may use such elements instead of paragraph markup for various reasons. (End of div.)

> This is a *block quotation* containing a single paragraph. Well, not quite, since this is not *really* quoted text, but I hope you understand the point. After all, this page does not use HTML markup very normally anyway.

The following contains links to the Comm-244 home page

Comm-244 Website, Week 3 page for class

Lists

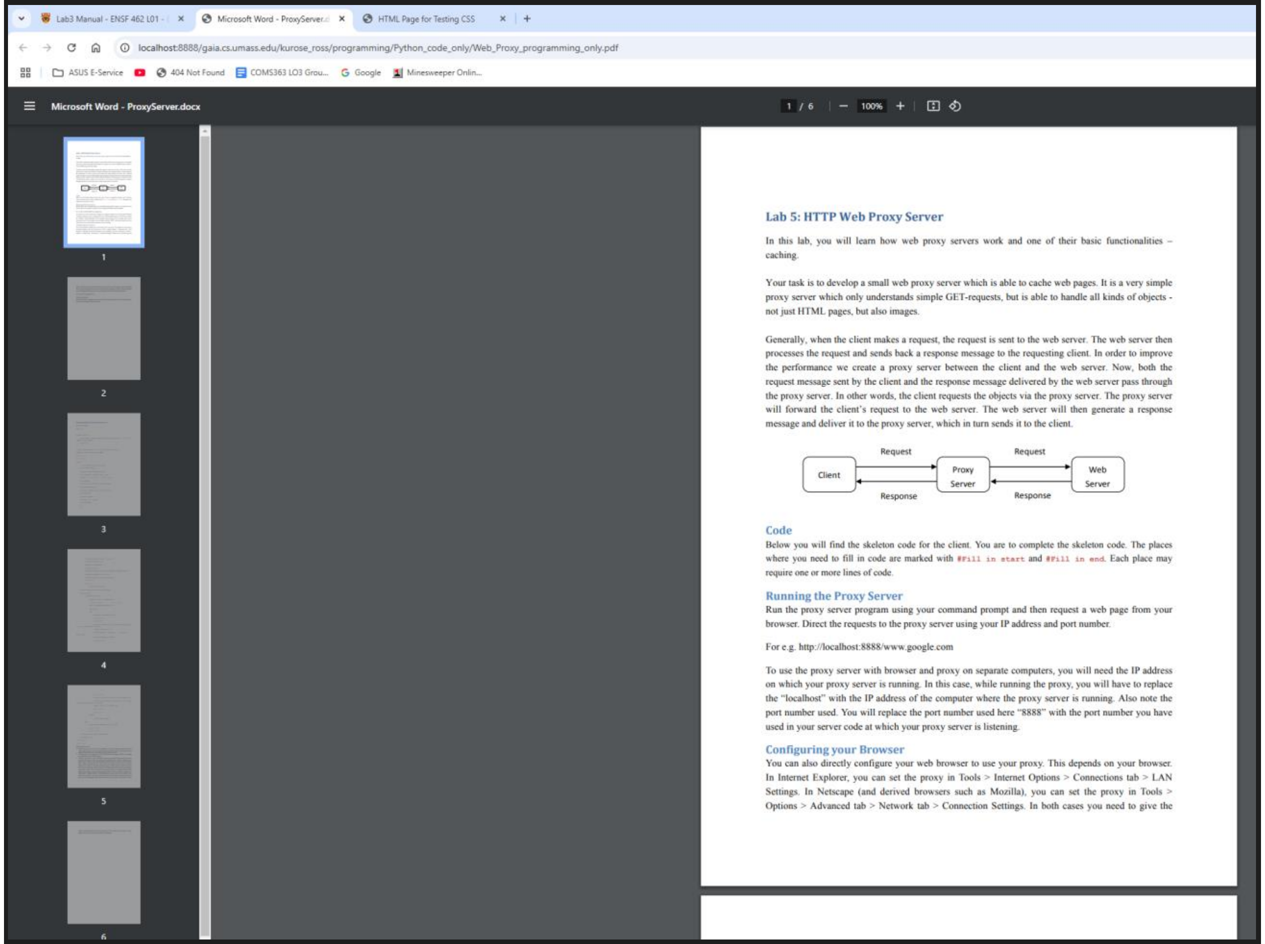
This is a paragraph before an **unordered** list (ul). Note that the spacing between a paragraph and a list before or after that is hard to tune in a user style sheet. You can't guess which paragraphs are logically related to a list, e.g. as a "list header".

- One.
- Two.
- Three. Well, probably this list item should be longer. Note that for short items lists look better if they are compactly presented, whereas for long items, it would be better to have more vertical spacing between items.
- Four. This is the last item in this list Let us terminate the list now without making any more fuss about it.

This is a paragraph before a **ordered** list (ol). Note that the spacing between a paragraph and a list before or after that is hard to tune in a user style sheet. You can't guess which paragraphs are logically related to a list, e.g. as a "list header".

1. One.
2. Two.
3. Three. Well, probably this list item should be longer. Note that if items are short, lists look better if they are compactly presented, whereas for long items, it would be better to have more vertical spacing between items.
4. Four. This is the last item in this list. Let us terminate the list now without making any more fuss about it.

This is a paragraph before a **definition** list (dl). In principle, such a list should consist of *terms* and associated definitions. But many authors use dl elements for fancy "layout" things. Usually the effect is not *too* bad, if you design user style sheet rules for dl which are suitable for real definition lists.


Lab3 Manual - ENSF 462 L01
Microsoft Word - ProxyServer.d
HTML Page for Testing CSS
localhost:8888/gaia.cs.umass.edu/kurose_ross/programming/Python_code_only/Web_Proxy_programming_only.pdf
ASUS E-Service
404 Not Found
COMS363 LO3 Grou...
Google
Minesweeper Onlin...
Microsoft Word - ProxyServer.docx
1 / 6
100%
Request
Request
Client
Proxy Server
Web Server
Response
Response

Lab 5: HTTP Web Proxy Server

In this lab, you will learn how web proxy servers work and one of their basic functionalities – caching.

Your task is to develop a small web proxy server which is able to cache web pages. It is a very simple proxy server which only understands simple GET-requests, but is able to handle all kinds of objects - not just HTML pages, but also images.

Generally, when the client makes a request, the request is sent to the web server. The web server then processes the request and sends back a response message to the requesting client. In order to improve the performance we create a proxy server between the client and the web server. Now, both the request message sent by the client and the response message delivered by the web server pass through the proxy server. In other words, the client requests the objects via the proxy server. The proxy server will forward the client's request to the web server. The web server will then generate a response message and deliver it to the proxy server, which in turn sends it to the client.

Code

Below you will find the skeleton code for the client. You are to complete the skeleton code. The places where you need to fill in code are marked with `#Fill in start` and `#Fill in end`. Each place may require one or more lines of code.

Running the Proxy Server

Run the proxy server program using your command prompt and then request a web page from your browser. Direct the requests to the proxy server using your IP address and port number.

For e.g. http://localhost:8888/www.google.com

To use the proxy server with browser and proxy on separate computers, you will need the IP address on which your proxy server is running. In this case, while running the proxy, you will have to replace the "localhost" with the IP address of the computer where the proxy server is running. Also note the port number used. You will replace the port number used here "8888" with the port number you have used in your server code at which your proxy server is listening.

Configuring your Browser

You can also directly configure your web browser to use your proxy. This depends on your browser. In Internet Explorer, you can set the proxy in Tools > Internet Options > Connections tab > LAN Settings. In Netscape (and derived browsers such as Mozilla), you can set the proxy in Tools > Options > Advanced tab > Network tab > Connection Settings. In both cases you need to give the

```python
from socket import *
import os
```

```python
# Create a server socket, bind it to a port and start listening
proxySerSock = socket(AF_INET, SOCK_STREAM)
#fill in start.
proxySerSock.bind(('', 8888))
proxySerSock.listen(1)
#fill in end.



while 1:
    print('Ready to serve...')
    #accept connection from clients
    proxyCliSock, addr = proxySerSock.accept()
    print('Received a connection from:', addr)

    # get the http request from client
    message = proxyCliSock.recv(1024).decode()
    print(message)

    # if message is not a GET request send a response 400 Bad request to the
client
    # close the connection and go to the next iteration of while loop waiting for
another request
    # from the same or a different client
    if not message.startswith("GET"):
        print("message is not a GET")
        # fill in start.
        response = "HTTP/1.1 400 Bad Request\r\n\r\n"
        proxyCliSock.send(response.encode())
        proxyCliSock.close()
        # fill in end.
        continue

    # Extract the pathname(including the filename) and hostname from the given
message
    slashPlusUrl = message.split()[1]
    url = slashPlusUrl.partition("/")[2]
    # fill in start
    hostn = url.split('/')[0]
    pathname = url.partition('/')[2]
    # fill in end.
    pathname = "/" + pathname
    # remove "www." from the hostname if it starts with "www."
    if hostn.startswith("www."):
        hostn = hostn.replace("www.", "", 1)

    print("pathname: " , pathname)
    print("hostname: ", hostn)

    fileExist = "false"

    directory = "./" + url

    try:
        # Check whether the file exists in the cache using open() method
        # If file exists it opens file and reads otherwise it throws an exception
        f = open(directory, "rb")
        object = f.read()

        # Send http response header and object
        response = "HTTP/1.1 200 OK\r\n\r\n"
        proxyCliSock.send(response.encode())
        proxyCliSock.send(object)
        f.close()

        fileExist = "true"
        print('Read from cache')
        #close socket and file
        #fill in start
        proxyCliSock.close()
        f.close()
        #fill in end

    except IOError: # Error handling for file not found in cache

        if fileExist == "false":
            # Create a socket on the proxyserver to connect to the original
server on port 80
            proxyAsClientSocket = socket(AF_INET, SOCK_STREAM)
            # Connect to the original server on port 80
            proxyAsClientSocket.connect((hostn, 80))



            # Create a GET request message and send the message to the server
using the socket just created in above lines
            # Hint: use pathname and hostn in the request message
            # fill in start
            request_line = f"GET {pathname} HTTP/1.1\r\n"
            host_header = f"Host: {hostn}\r\n"
            connection_header = "Connection: close\r\n\r\n"
            get_request = request_line + host_header + connection_header
            proxyAsClientSocket.send(get_request.encode())
            # fill in end

            # initialize response to empty in binary format - works for any type
of document
            total_response = b''

            # receive data from web server
            #Hint: You can use a while loop and get response in chunks until it
is finished.
            #Fill in start
            while True:
                response_chunk = proxyAsClientSocket.recv(4096)
                if not response_chunk:
                    break
                total_response += response_chunk
            #Fill in end



            # Separate header and object
            # Hint: use split function. Check the lecture notes to see what
separates the response header and object
            response_header, response_object = total_response.split(b'\r\n\r\n',
1)



            if b'200 OK' in response_header:
                # if the response is a 200 OK response create the directory and
file and write the object into the file
                # Then, send http response header and object to the client
                #Fill in start
                os.makedirs(os.path.dirname(directory), exist_ok=True)
                with open(directory, "wb") as cache_file:
                    cache_file.write(response_object)

                proxyCliSock.send(response_header + b'\r\n\r\n' +
response_object)
```

```python
                #Fill in end
            else:
                #Otherwise, i.e., if response is not a 200 OK message,send 400
bad response
                #Fill in start
                response = "HTTP/1.1 400 Bad Request\r\n\r\n"
                proxyCliSock.send(response.encode())
                #Fill in end

            # close socket between proxy and origin server
            proxyAsClientSocket.close()
        else:
            break
    except:
        print("An Exception Occurred")
    finally:
        # close socket between proxy and client
        proxyCliSock.close()
    break #to test file with multiple objects. If the tested URLs have only one
object you can remove "break"


#close the main proxy listening socket
proxySerSock.close()
print("finish")
```