ICN 2022

HW1 report

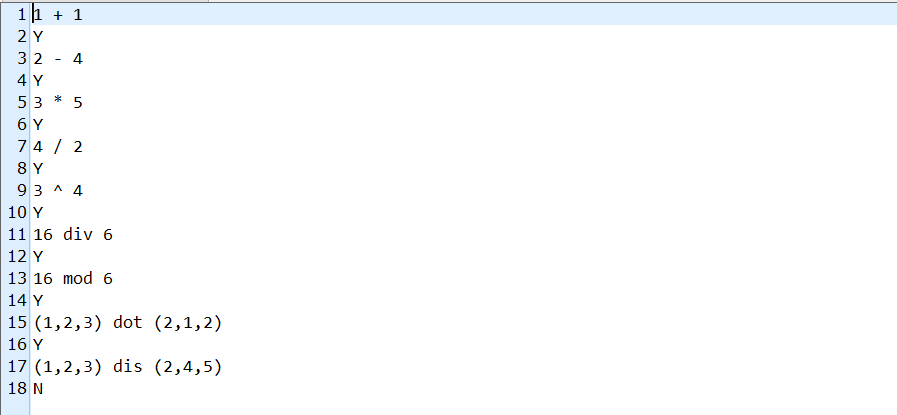
B08901002 林宸漢

1.

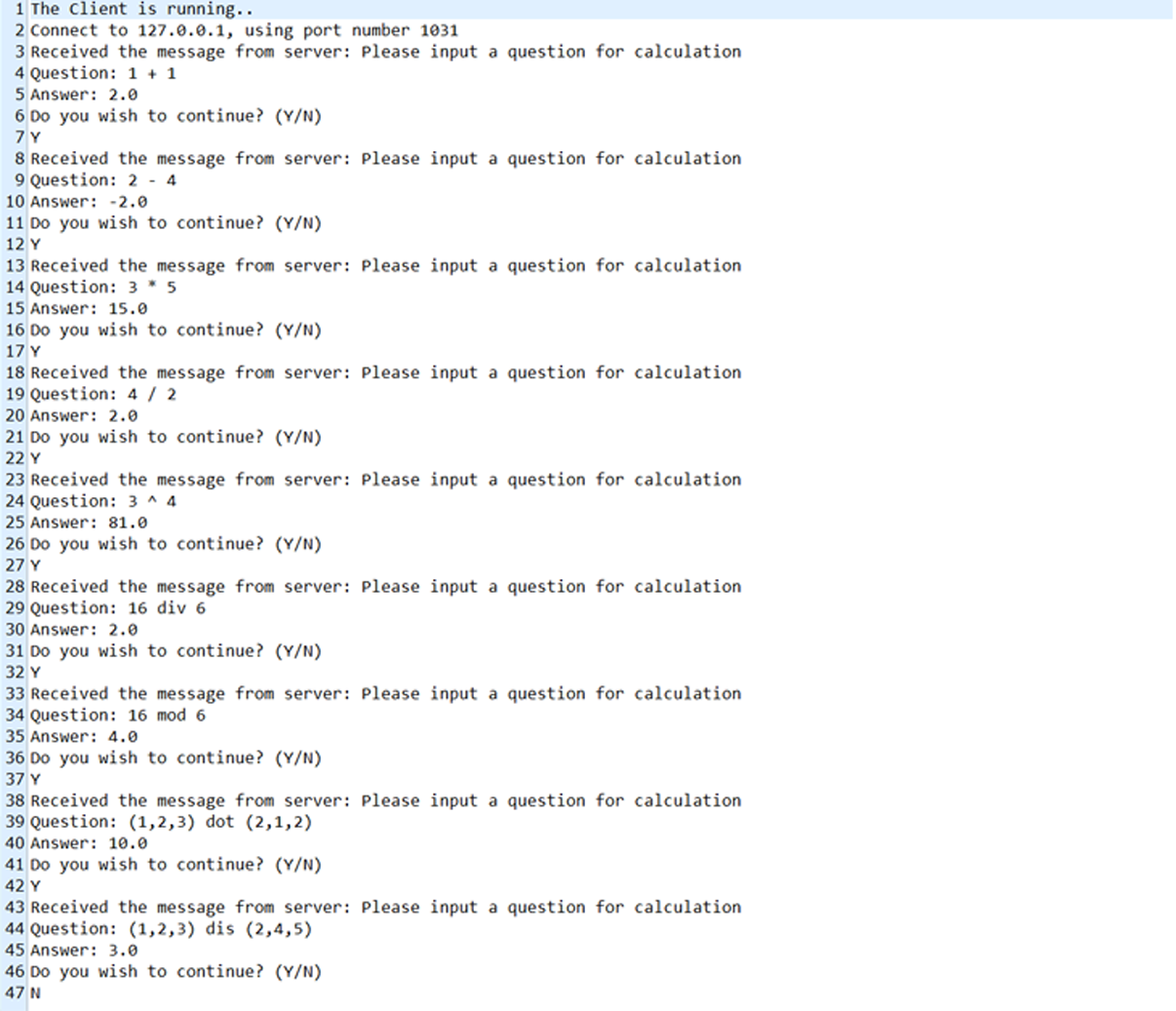
About the bonus part of p1, I have done the 5 additional functions. These functions are as below:

|  |  |  |  |
| --- | --- | --- | --- |
| function | A ^ B | A div B | A mod B |
| explanation | Calculate A to the power of B | Calculate the quotient of A/B | Calculate the remainder of A/B |
| Sample Input | 3 ^ 4 | 16 div 6 | 16 div 6 |
| Sample output | 81.0 | 2.0 | 4.0 |
| function | **A dot B** | **A dis B** |
| explanation | Calculate the inner product of A and B | Calculate the Euclidian distance of A and B |
| Sample Input | (1,2,3) dot (2,1,2) | (1,2,3) dis (2,4,5) |
| Sample output | 10.0 | 3.0 |

If we revise the “p1\_testcase” as below:



We could get the revised “p1\_testcase\_golden” as below:



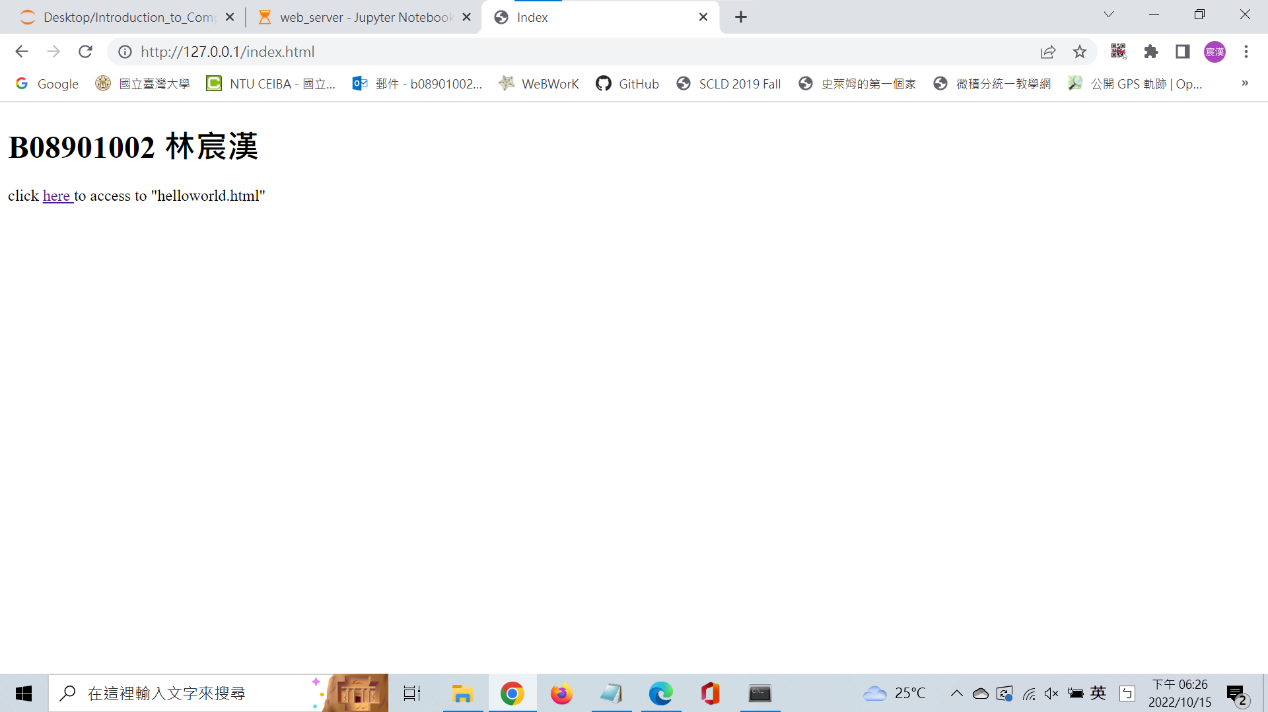
As above, we have passed TA’s test case and our own test cases of 5 bonus math functions.

2.(a)

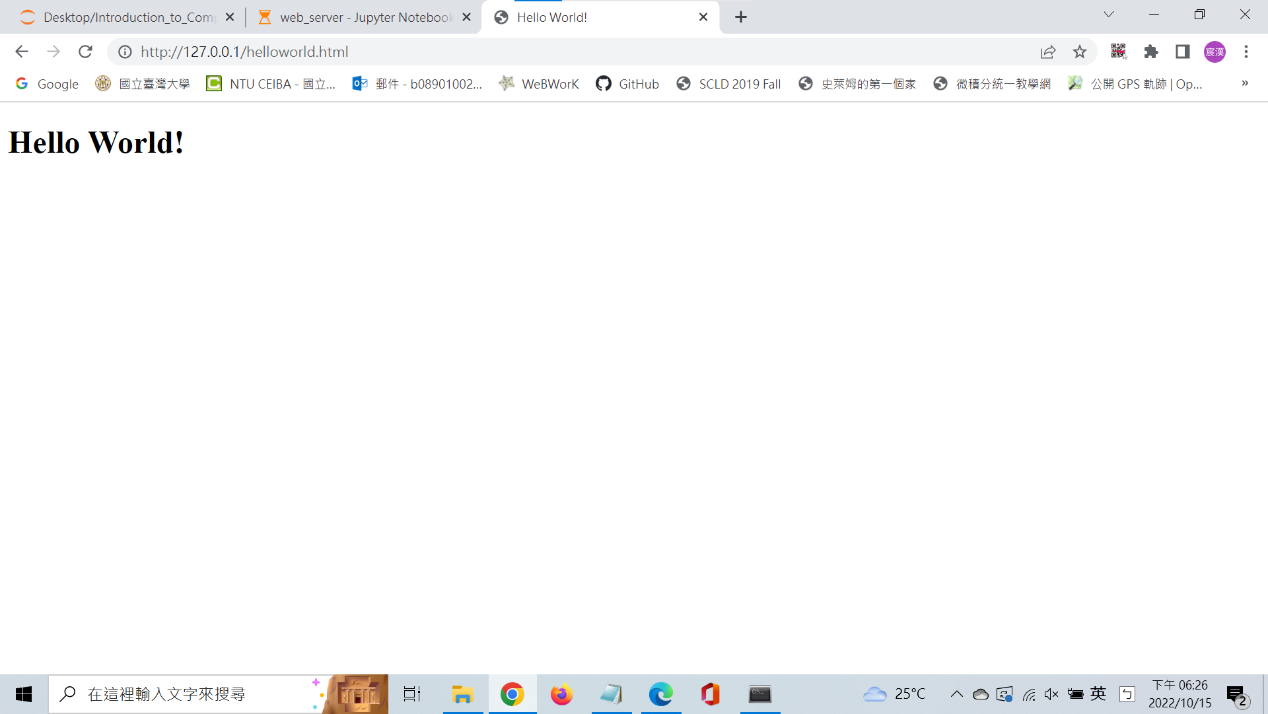
Below are the output results of p2:

First, we test codes on the **local machine**. We use **chrome** as our browser.

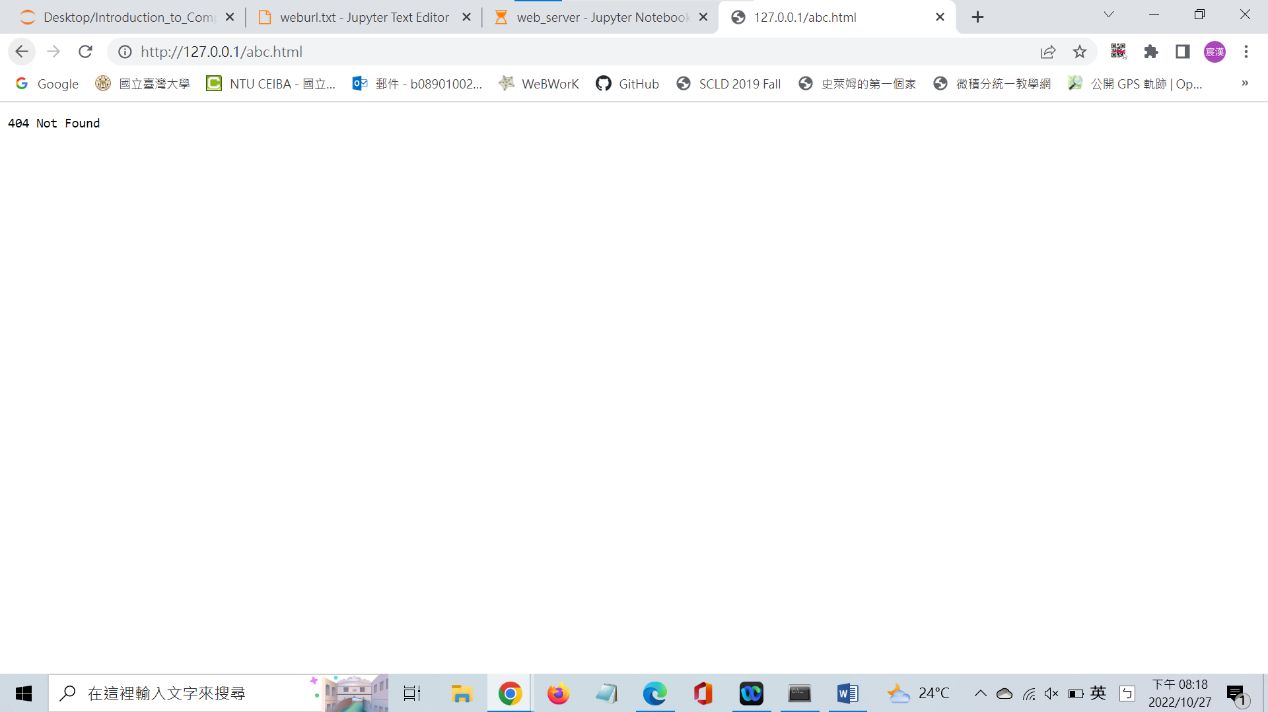
Below is the result that we try to request “index.html”.



Next, we click the hyperlink on the website and get the result below:

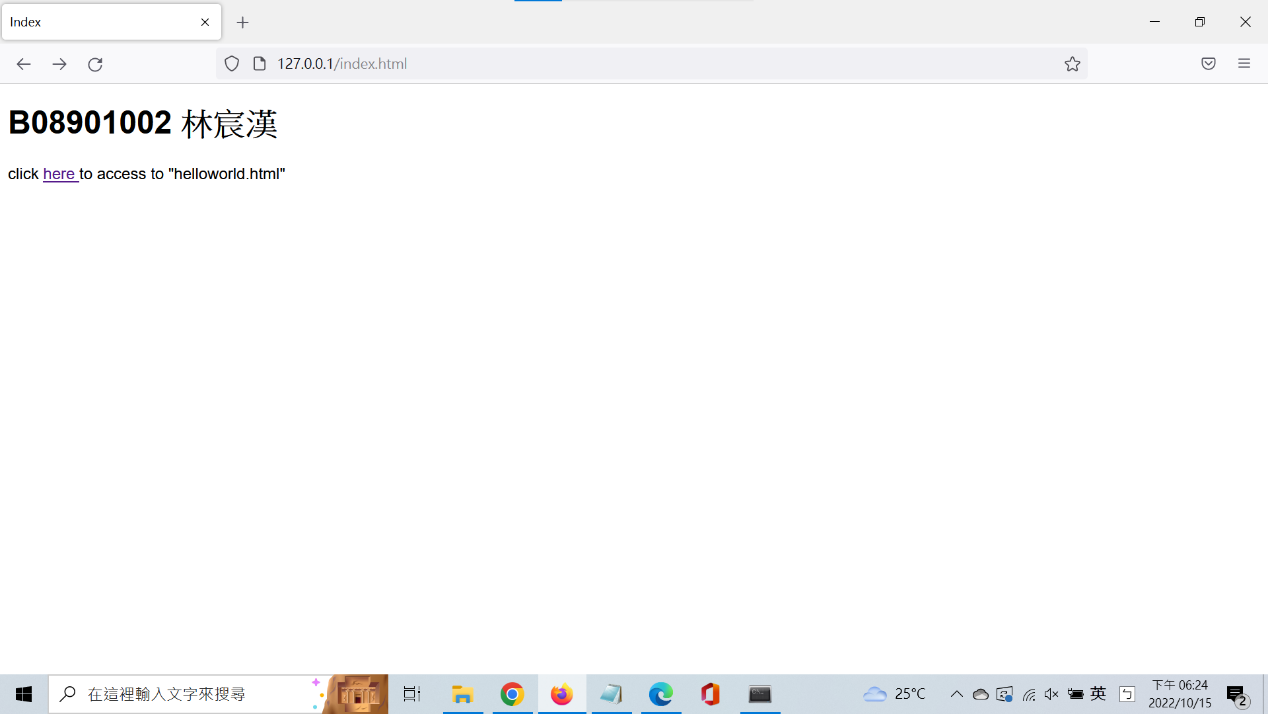


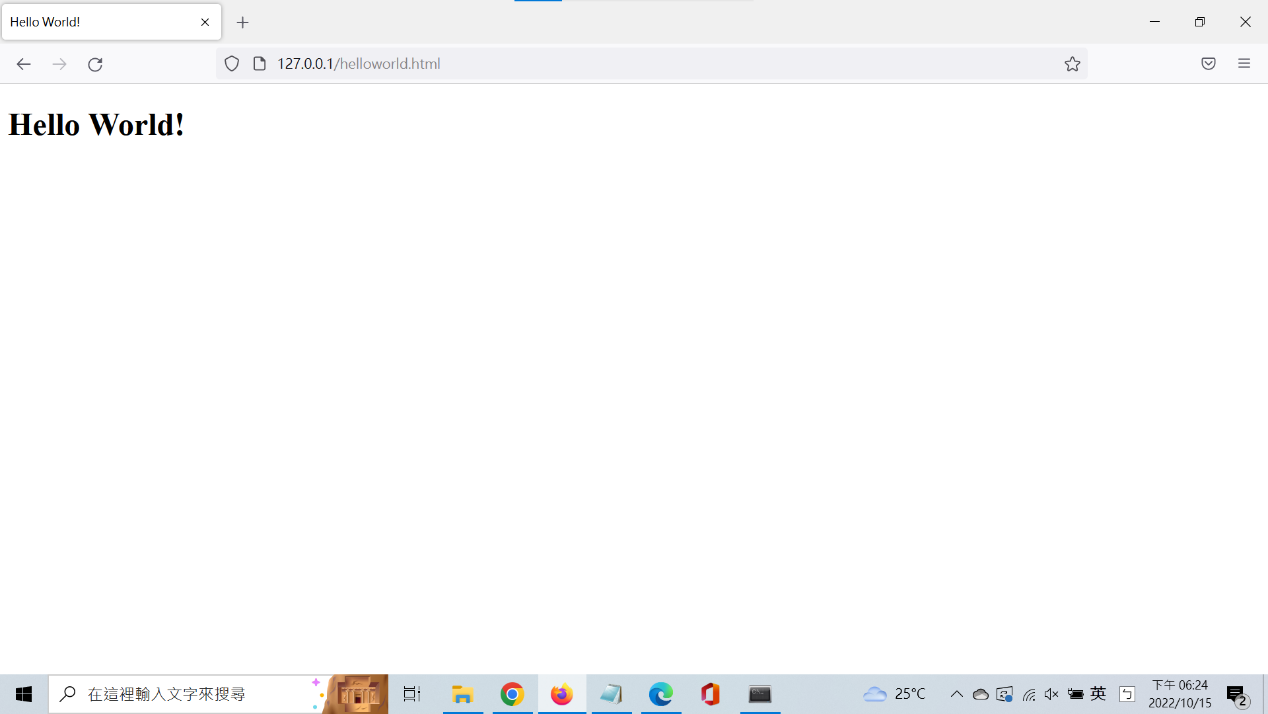
Finally, we try to request “abc.html”, which doesn’t exist in our server. The result is as below:

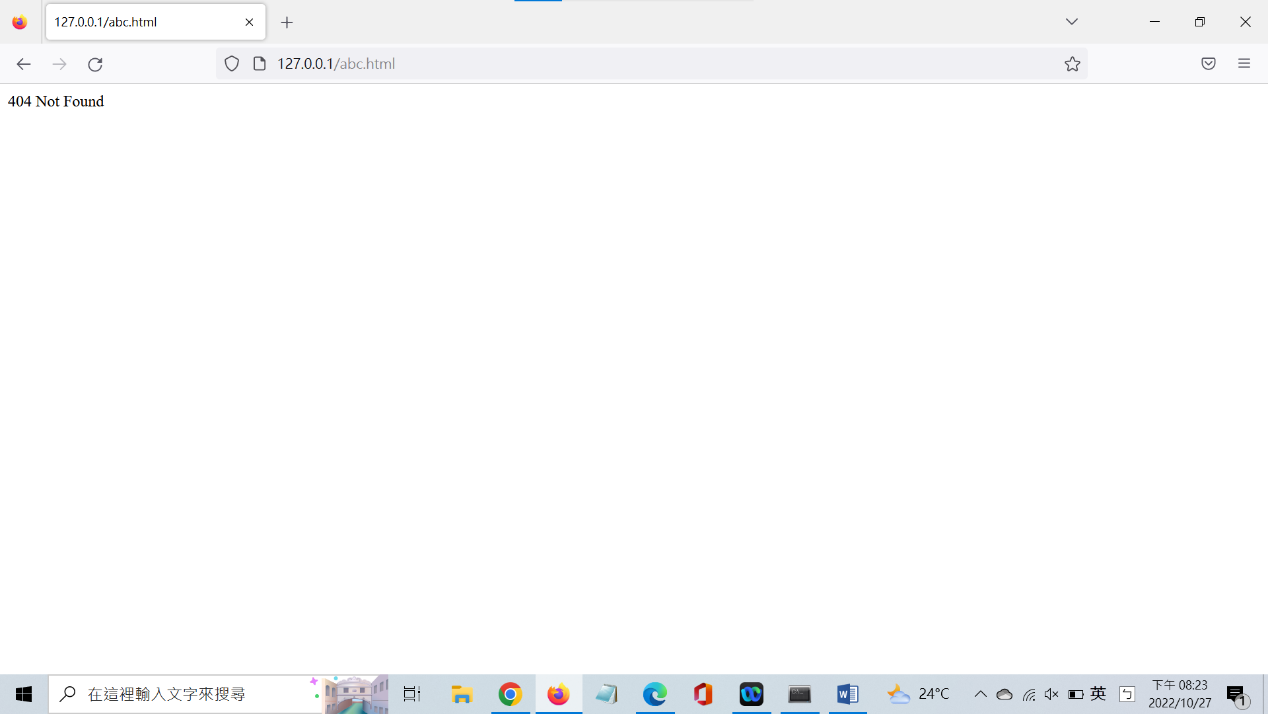


As above, it seems that we accomplish the assignments in p2.

Then we try another browser. We change to **Firefox** as our browser. Following the similar steps, we get the results as below:







2(b)

There are totally 8 TODO parts. We follow TA’s hint and complete these parts as below:

* # Create a socket and bind the socket to the address

Referring to the example in Ch2 slides p.38, we use python’s package: **socket** to create a socket on the web server. Then we use **.bind()** to bind the socket to the address. Here, we use **HOST, PORT = "127.0.0.1", 80**.

* # Establish the connection

Referring to the example in Ch2 slides p.39, we use **.accept()** to accept a new connection (request) and create a new socket for the connection.

* # Receive a HTTP request from the client

Referring to the example in Ch2 slides p.39, we use **.recv(1024)** to receive data by socket and use **.decode()** todecode the message to string.

* # Read data from the file that the client requested

# Split the data into lines for further transmission

Since we already have **f** opened in the sample code, we use **.read()** to get the contents(/data) of f. Then we use **.split()** to split the data into lines.

* # Send HTTP Status to the client

Referring to the example in Ch2 slides p.21, we use **.send()** to send HTTP Status to the client.

* # Send the Content Type to the client

Referring to the example in Ch2 slides p.21, we use **.send()** to send the client the Content Type of requested data.

* # Send the response message if the file is not found

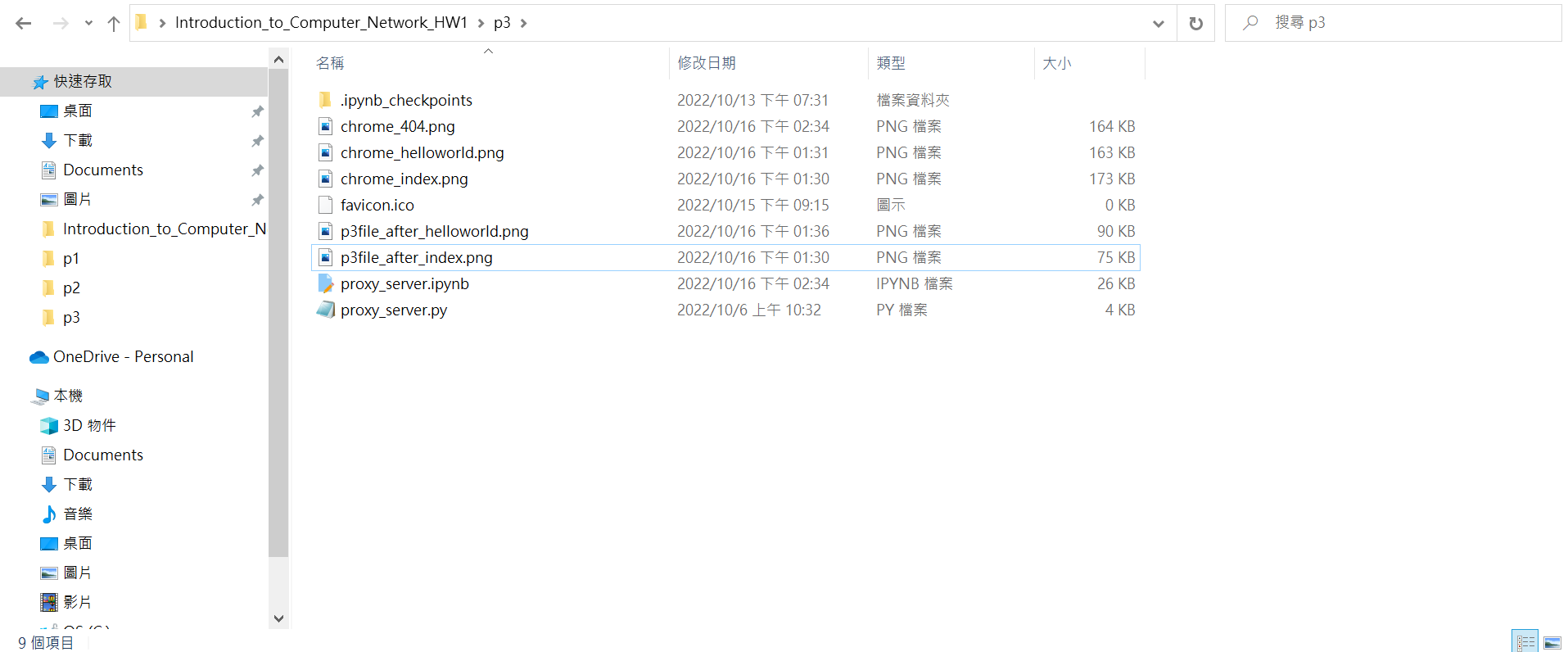
If the file does not exist, we need to send an error message to the client. We use **.send()** to send this message to client.

* # Close client socket

Referring to the example in Ch2 slides p.39, we use **.close()** to close the client socket.

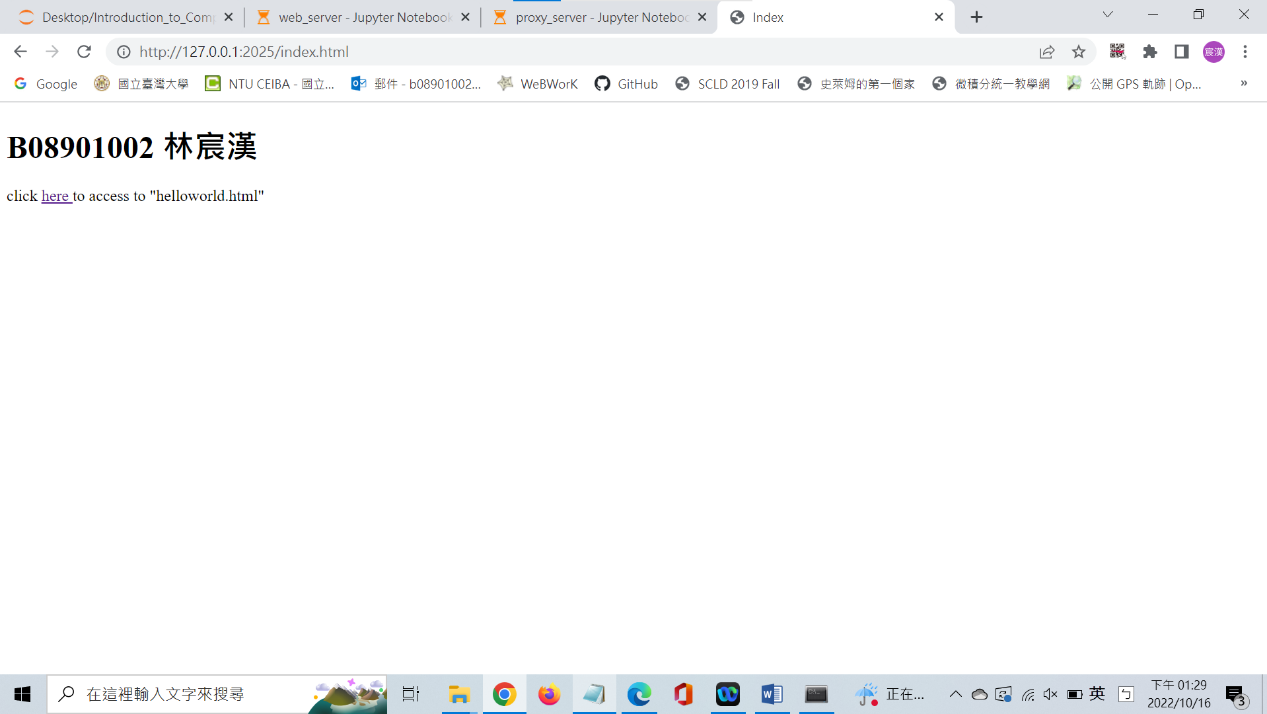
3(a)

As explained in “ICN2022\_HW#1”, there isn’t any .html file under the proxy server.



As above, initially, there isn’t any .html file.

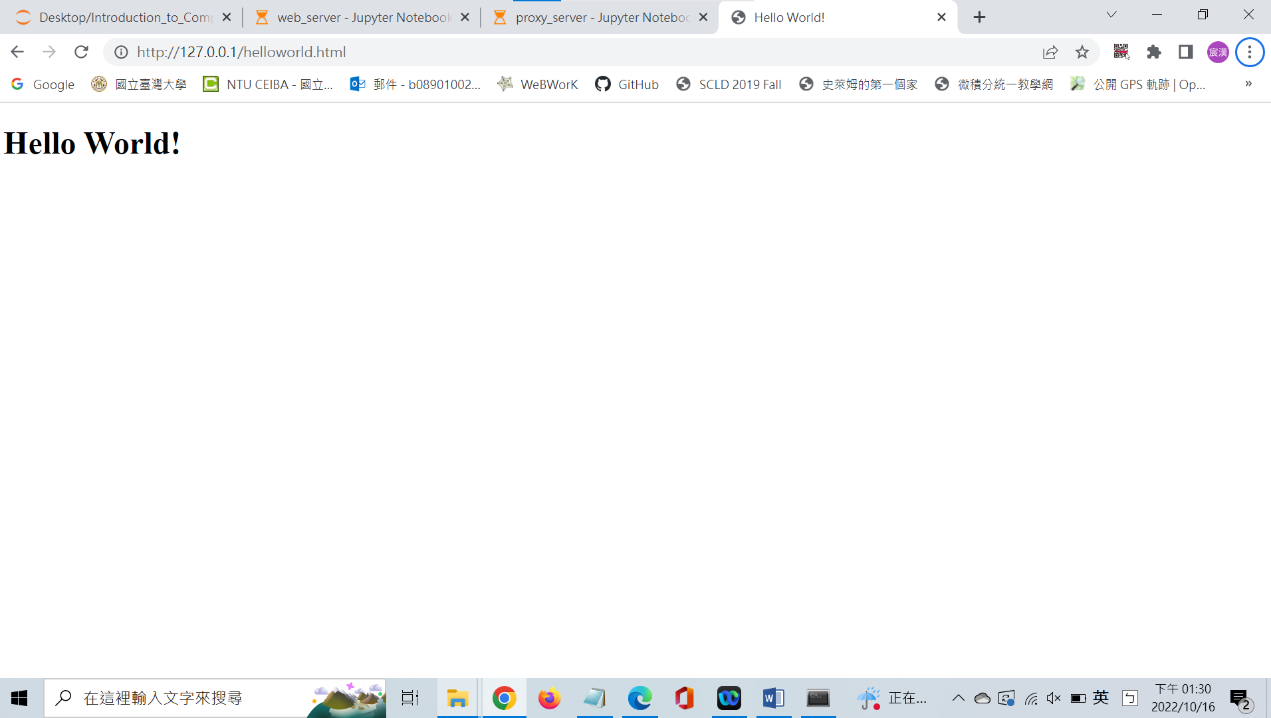
Next, we ask the proxy server(using **port=2025**) to request “index.html”. And we get the results as below:

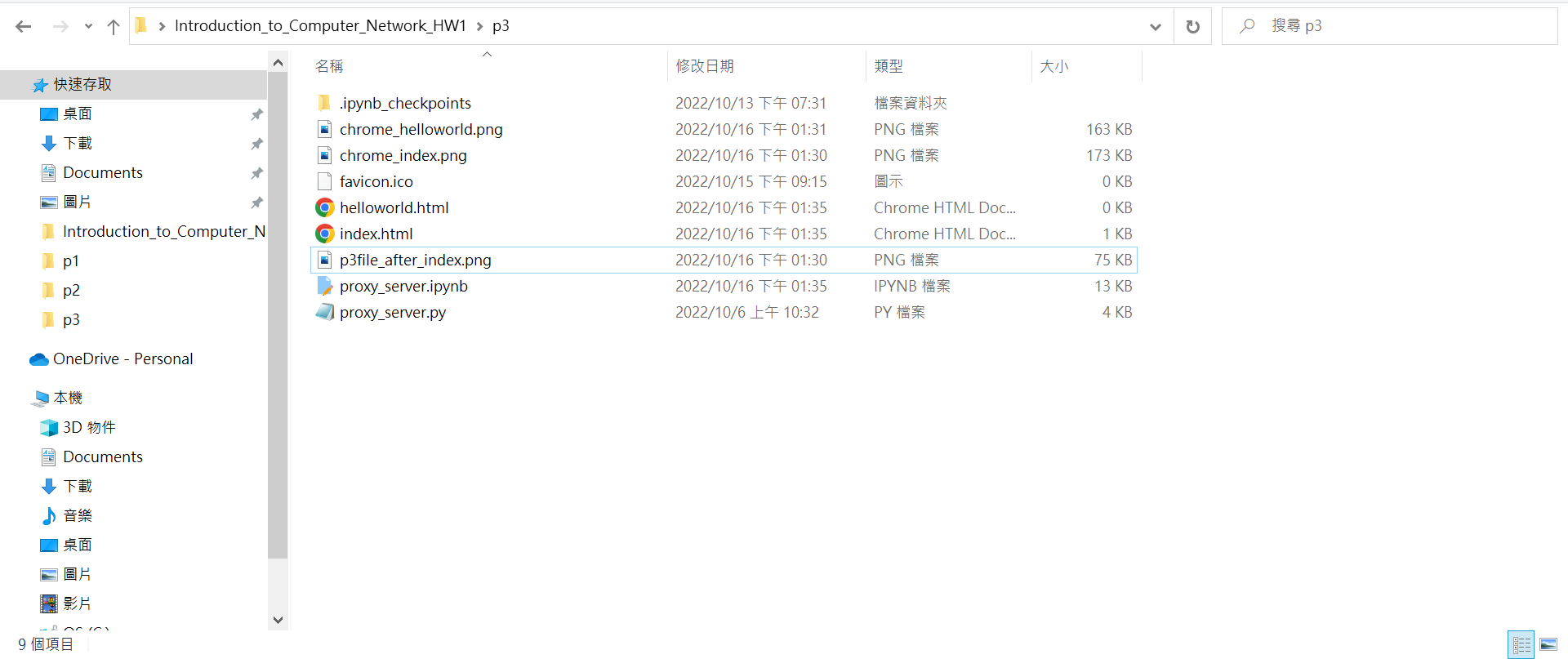




As above, we successfully see the index.html. Moreover, we can see the “index.html” appears under the file, which is what we called **“caching”**.

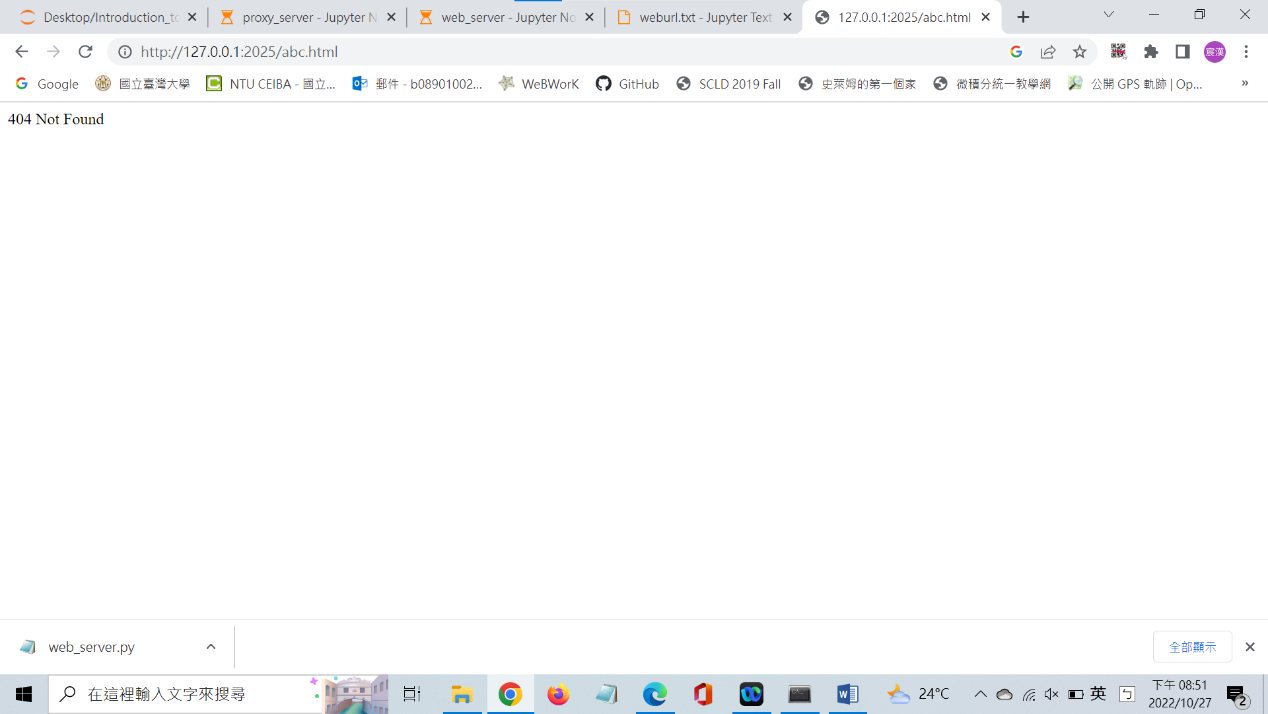
Similar to p2, then we click the hyperlink on the website to request “helloworld.html”. And we get the results as below:





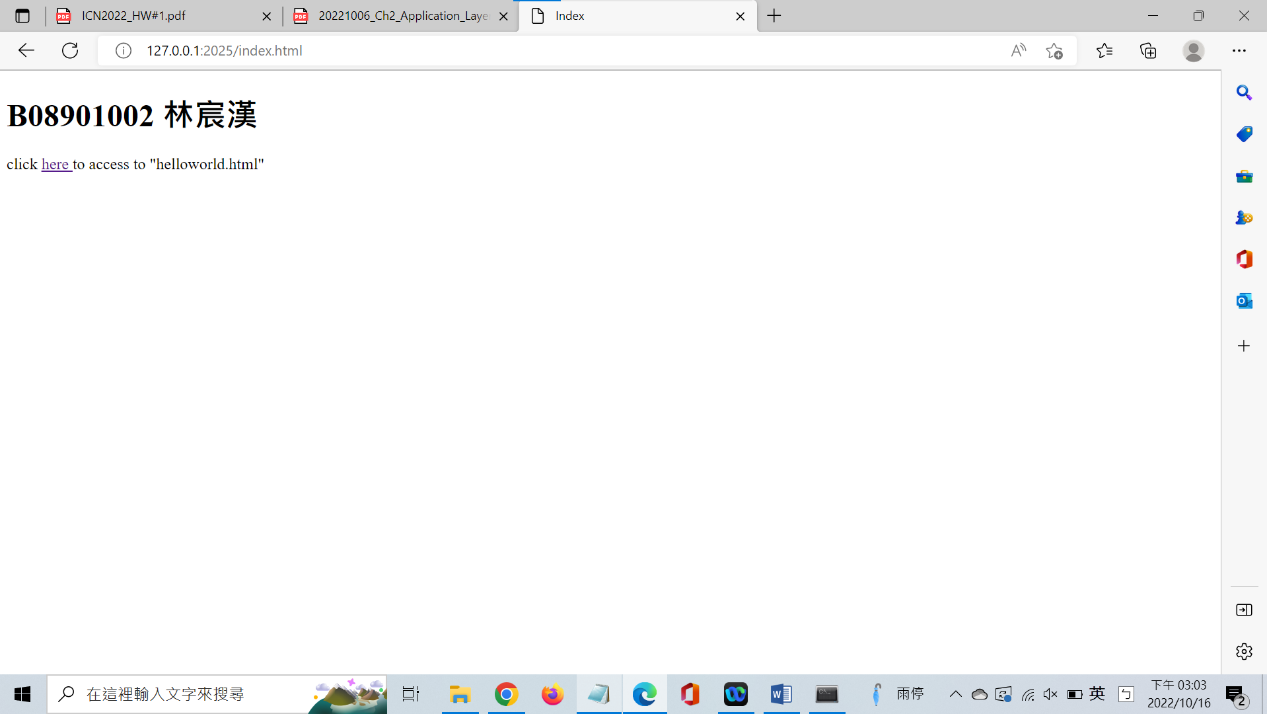
As above, we successfully see the helloworld.html. Moreover, we can see the “helloworld.html” appears under the file. We still successfully cache the file.

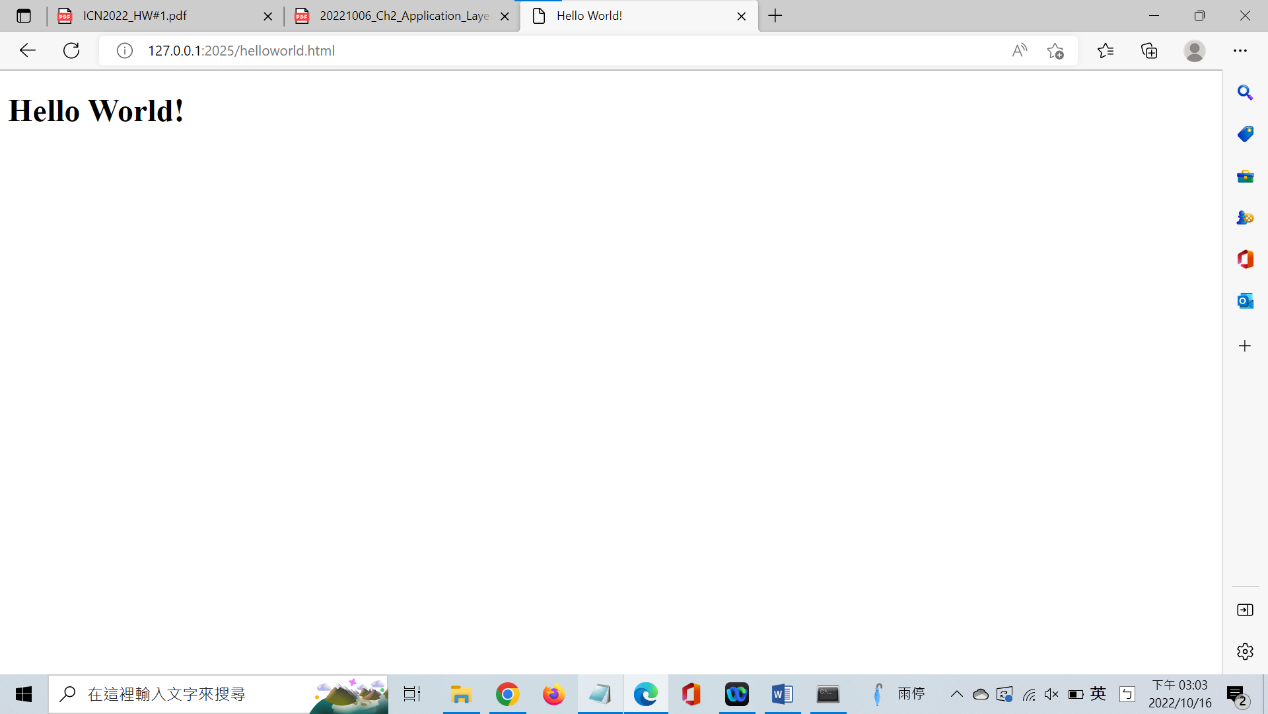
Finally, we try to ask the proxy server to request “abc.html”, which exists neither in the proxy server nor in the web server. And we get the results as below:

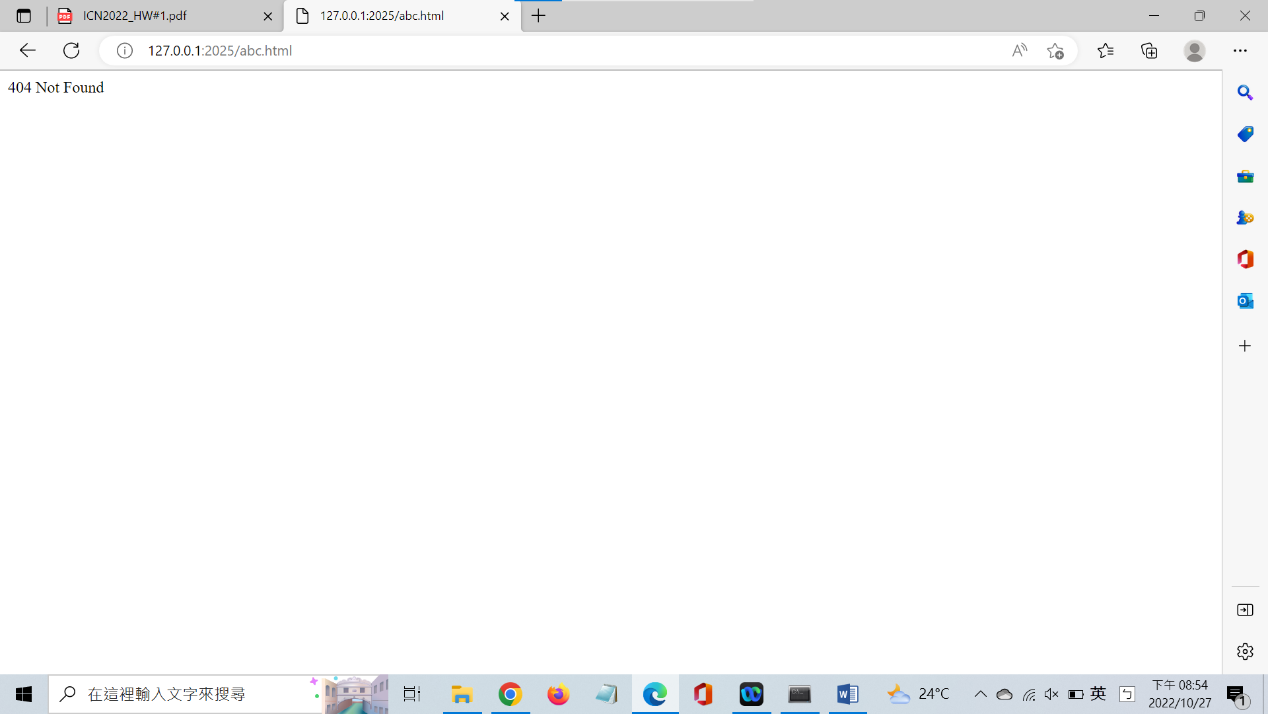


As above, we successfully send the “404 not found” message to the client(browser).

Then we try another browser. We change to **Microsoft Edge** as our browser. Following similar steps, we get the results as below:







As above, we can successfully access HTML files through another browser.

In addition, the HTML files are still cached under the p3 file, just as the result of the chrome case. Since they look changed nothing as in the chrome case, I ignore the screenshots of them in the report.

3(b)

There are totally 10 TODO parts. However, some of them are the same as the TODO parts in p2. So we only discuss the different parts from p2 here. We follow TA’s hint and complete these parts as below:

* # Create a server socket, bind it to a port and start listening

Here, we bind “TCPServerSocket” with **HOST, PORT = "127.0.0.1",2025**. And we use **.listen(1)** to make the proxy server start listening.

* # Proxy Server finds the file (cache hit) and generates a response message

# Send the file back to the client

Referring to the sample code in p2, at first, we split “DataInFile” into lines. For each line, we send it to “TCPClientSocket”, while **we need to send a “ “(space) between each line** so that the texts in the website won’t bunch together. Then, we send **“\r\n”** as a signal that we finish the sending. Finally, we close the “TCPClientSocket”.

(start Error handling part)

* # Connect the socket to the web server port

Here, we make SocketOnProxyServer connect to **HOST, PORT = "127.0.0.1",80**, Which make connect to web server in p2.

* # Read the response into buffer

Here, the proxy server doesn’t have the requested file and it asks the web server to request the file. When the request is sent, if the file exists, the web server will send the data back to the proxy server. Hence, the proxy server will read the web server’s replies and save them into a buffer, called **“DataInFile”** in my code.

Meanwhile, if the file doesn’t exist, the proxy server will receive a “404 not found” error message. Then, it would **convey the error message to the client and finish this request**(While loop).

In this part, I observe that **in p2, we can directly send “404 not found” to the Client without sending “HTTP status” code and “Content-Type” header line before, while we can’t do the same thing in p3**. The status code and header line is needed before we send the error message to Client.

I infer that the reason is at **their different port**. The web server has **port = 80**, which is the default port of HTTP and is open in the firewall system at most of the time. Hence, the status code and header line are not needed. As for the proxy server, whose **port = 2025**, it doesn’t use a special port like the web server. Hence, HTTP may need more information to identify the data sent from port 2025.

* # Create a new copy in the cache for the requested file

# Also send the response back to Client socket and the corresponding file from cache

Before we send data, we first need to **send the status code and header line** to Client.

Next, for each line in **“DataInFile”**, we **write the line into the copy file and send it to Client**.

Noted that when we copy the “index.html” from the web server at the proxy server, we need to change the web URL in “index.html”. That is, we have “<a href="http://127.0.0.1:80/helloworld.html">” received from the web server and we need to **change** **“80” to “2025”**. If we forget to do this, **the browser will use the web server when requesting “helloworld.html”**, which is not what we want.