Lab session Overview

CS341 Fall 2020

cs341-ta@cds.kaist.ac.kr

2020/09/03

Other course workload

- Quiz
  - Approximately 10~12 short online quizzes.
  - Consists of 3~5 short multiple choice questions.
  - Every Thursday 9:00 am~9:05am.
  - You can check the answers on KLMS's notice.
- HW Assignment (Textbook questions, wireshark)
  - 8~10 homework assignments (both).
  - 1~2 weeks for each assignment.
  - PDF format. English writing.
- Reading Assignment (Essay)
  - 6~8 essay writing throughout this course.
  - 1~2 weeks for each assignment.
  - PDF format. English writing.
- Plagiarism and cheating are serious offenses and may be punished by failure on assignments; failure in course;

Overview

6 Lab Sessions (Schedule may change, check KLMS) + QnA

- 9/3 (Simple Web Communication)
- 9/17 (TCP Socket)
- 10/8 (NS3 - TCP Congestion Control)
- 10/29 (NS3 - AODV)
- 11/12 (NS3 - BGP)
- 11/26 (NS3 - Wireless)

2~3 weeks for each project (※ No KENS this semester)

Programming Language & Platform

- Practice #1: Python 3
- Practice #2~6: C++ (gcc, g++)
  - **You should study C++ from now on** if you are not familiar with.
- Practice #3~6: Network Simulator 3
  - We will give some tutorials to install and run NS3.

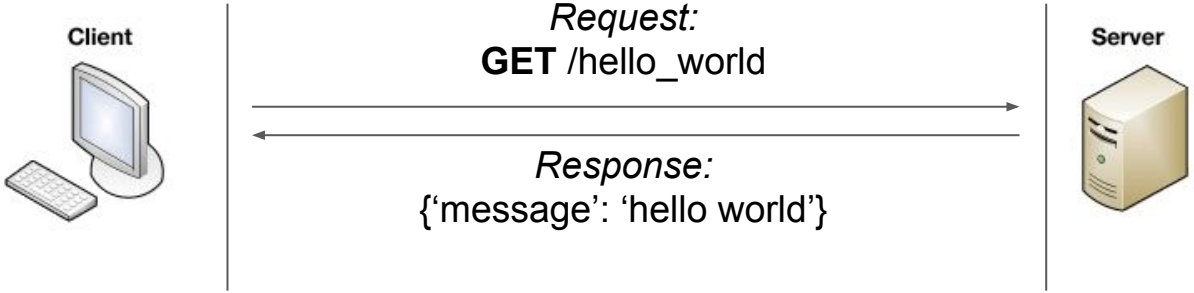
From Practice #2, we recommend you to install Ubuntu on VM, or ask for cloud server from us (we will notice about cloud server later).

Practice #1:
Simple Web Communication

Sumin Han (hsm6911@kaist.ac.kr)
2020/09/03

Goal

- Practice simple web communication between server and client.



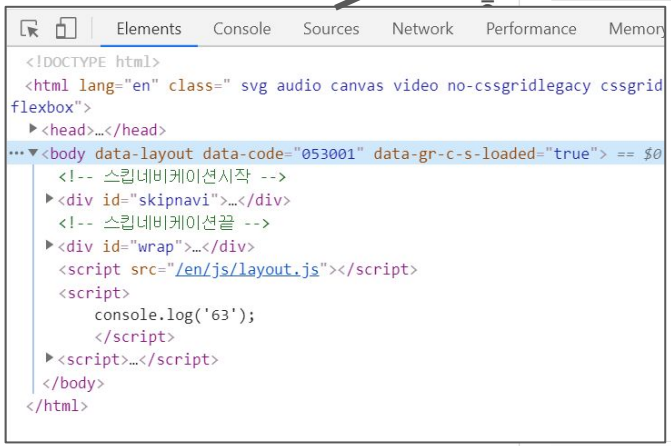
Background: GET

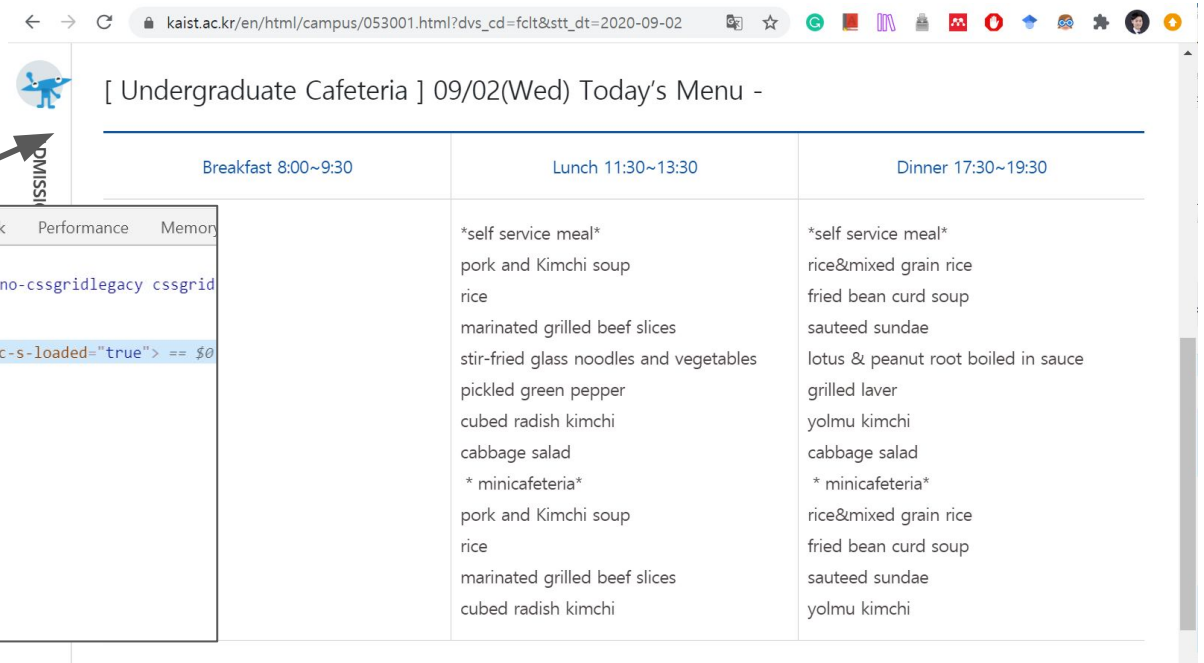
parameters = {
dvs_cd=fclt,
stt_dt=2020-09-02
}

- Example: https://www.kaist.ac.kr/en/html/campus/053001.html?dvs_cd=fclt&stt_dt=2020-09-02

browser interprets html file

server sends
some '.html' file.

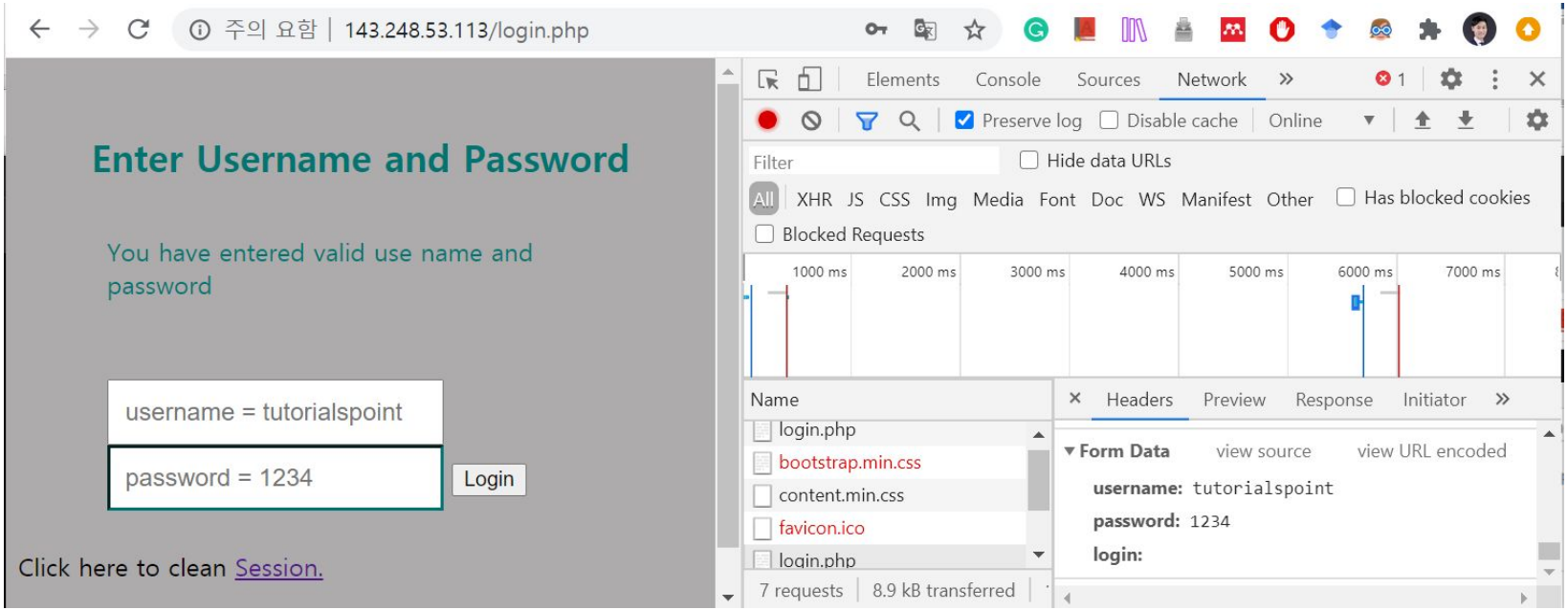




Background: POST

- https://www.edureka.co/blog/get-and-post-method/
- http://143.248.53.113/login.php

- It appends form-data to the body of the HTTP request in such a way that data is not shown in the URL.



Background: JSON format

- Just like sending html file, you can also send JSON data.
- You can create a JSON string from Python dictionary simply:

```
import json
data = {'key': 'value'}
print(json.dumps(data))
```

```
{"key": "value"}
```

Example [edit]

The following example shows a possible JSON representation describing a person.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Tip: SHA256

- A cryptographic hash algorithm: https://en.wikipedia.org/wiki/SHA-2

```
import hashlib
text = 'summer'
print(hashlib.sha256(text.encode()).hexdigest())
```

```
e83664255c6963e962bb20f9fcfaad1b570ddf5da69f5444ed37e5260f3ef689
```

Tip: Collatz number

The following iterative sequence is defined for the set of positive integers:

$n \to n/2$ (n is even)
$n \to 3n + 1$ (n is odd)

Using the rule above and starting with 13, we generate the following sequence:

$$13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

TODO (1/3)


1
Request message from
/hello_world, and print message
Client
Server
Request:
GET /hello_world
Response:
{‘message’: ‘hello world’}

TODO (2/3)

2 Request /hash for SHA256 hashed value

GET /hash?name=summer

{'result': 'e83664255c6963e962bb20f9...'}

3 Request /collatz for next collatz number

POST /collatz

{'name': 'summer',
'hash': 'e83664255c...',
'number': 3}

{'result': 10}

Repeat ② - until the result is 1

POST /collatz

{'name': 'summer',
'hash': 'e83664255c...',
'number': 10}

{'result': 5}

Collatz number:

$n \rightarrow n/2$ (n is even)

$n \rightarrow 3n + 1$ (n is odd)

TODO (3/3)

E1

When the hashed value of name and the hash input are not matched:

POST /collatz

{‘name’: ‘summer’,
‘hash’: ‘wrong_hash...’,
‘number’: 3}

{‘error’: ‘HASH NOT MATCHED’}

E2

When the number input is not digits:

POST /collatz

{‘name’: ‘summer’,
‘hash’: ‘e83664255c...’,
‘number’: ‘13.54’}

{‘error’: ‘NUMBER NOT INTEGER’}

(it is not mandatory to follow skeleton)

Skeleton (*Use Flask: pip install flask*)

server.py

```python
from flask import Flask, render_template, request
import json, hashlib

app = Flask(__name__)

@app.route('/hello_world', methods=['GET'])
def route_hello_world():
    return json.dumps({'message': 'hello world'})


@app.route('/hash', methods=['GET'])
def route_hash():
    return json.dumps({'result': 'HASHED_VALUE'})

@app.route('/collatz', methods=['POST'])
def route_collatz():
    return json.dumps({'result': 'NEXT_COLLATZ_NUMBER'})

if __name__ == '__main__':
    app.run()
```

skeleton_client.py

```python
import sys

url = sys.argv[1]
name = sys.argv[2]
number = int(sys.argv[3])

```

Search how to deal with parameters on the server side for GET and POST methods using Flask.

Requirement - client.py

$ python client.py **(1)url** **(2)text** **(3)int**

```
C:\>python client.py "http://143.248.53.113:5000" summer 5
hello world
e83664255c6963e962bb20f9fcfaad1b570ddf5da69f5444ed37e5260f3ef689
16
8
4
2
1
```

1. Print hello world message.
2. Print SHA256 hashed value of input text.
3. Print next Collatz number of input, and repeat until it produces 1.

Requirement - server.py

```
C:\>python server.py
 * Serving Flask app "server" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [03/Sep/2020 15:48:40] "GET /hello_world HTTP/1.1" 200 -
127.0.0.1 - - [03/Sep/2020 15:48:42] "GET /hash?name=summer HTTP/1.1" 200 -
127.0.0.1 - - [03/Sep/2020 15:48:44] "POST /collatz HTTP/1.1" 200 -
127.0.0.1 - - [03/Sep/2020 15:48:46] "POST /collatz HTTP/1.1" 200 -
127.0.0.1 - - [03/Sep/2020 15:48:48] "POST /collatz HTTP/1.1" 200 -
127.0.0.1 - - [03/Sep/2020 15:48:50] "POST /collatz HTTP/1.1" 200 -
127.0.0.1 - - [03/Sep/2020 15:48:52] "POST /collatz HTTP/1.1" 200 -
```

Run server first, then test with client.py

run client.py on your localhost server

```
C:\>python client.py "http://localhost:5000" summer 5
hello world
e83664255c6963e962bb20f9fcfaad1b570ddf5da69f5444ed37e5260f3ef689
16
8
4
2
1
```

We will evaluate your code on Python 3.4+, and latest version of other libraries.

Scoring (50 pt.)

- client.py (15 pt.)
  - Sample Flask Server running on http://143.248.53.113:5000/
  - GET hello world message (3 pt.).
  - GET hash for a text, and print (5 pt.).
  - POST next Collatz number and print until the result is 1 (7 pt.).
- server.py (25 pt.)
  - Route function for /hello_world (1 pt.). *- already in the skeleton.*
  - Route function for /hash (8 pt.).
  - Route function for /collatz (16 pt.).
    - This includes error handling E1 (4 pt.) and E2 (4 pt.)
- Report in .pdf (10 pt.) - maximum 3 pages.
  - Screenshots of the results from executing server.py and client.py.
  - Explain your code implementation.

Submit a zip file containing:

- client.py
- server.py
- report.pdf

Due: 9/16, 11:59 pm

plagiarism,
late submission:
0 point