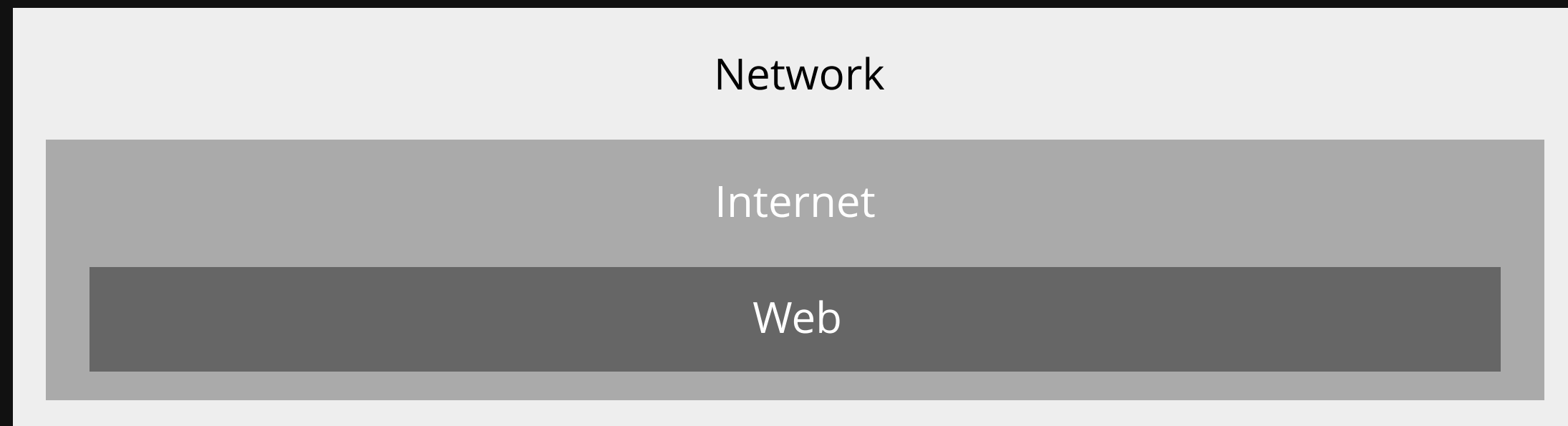


COMP1531

4.2 - Web - HTTP & Flask

Computer Networks



The network

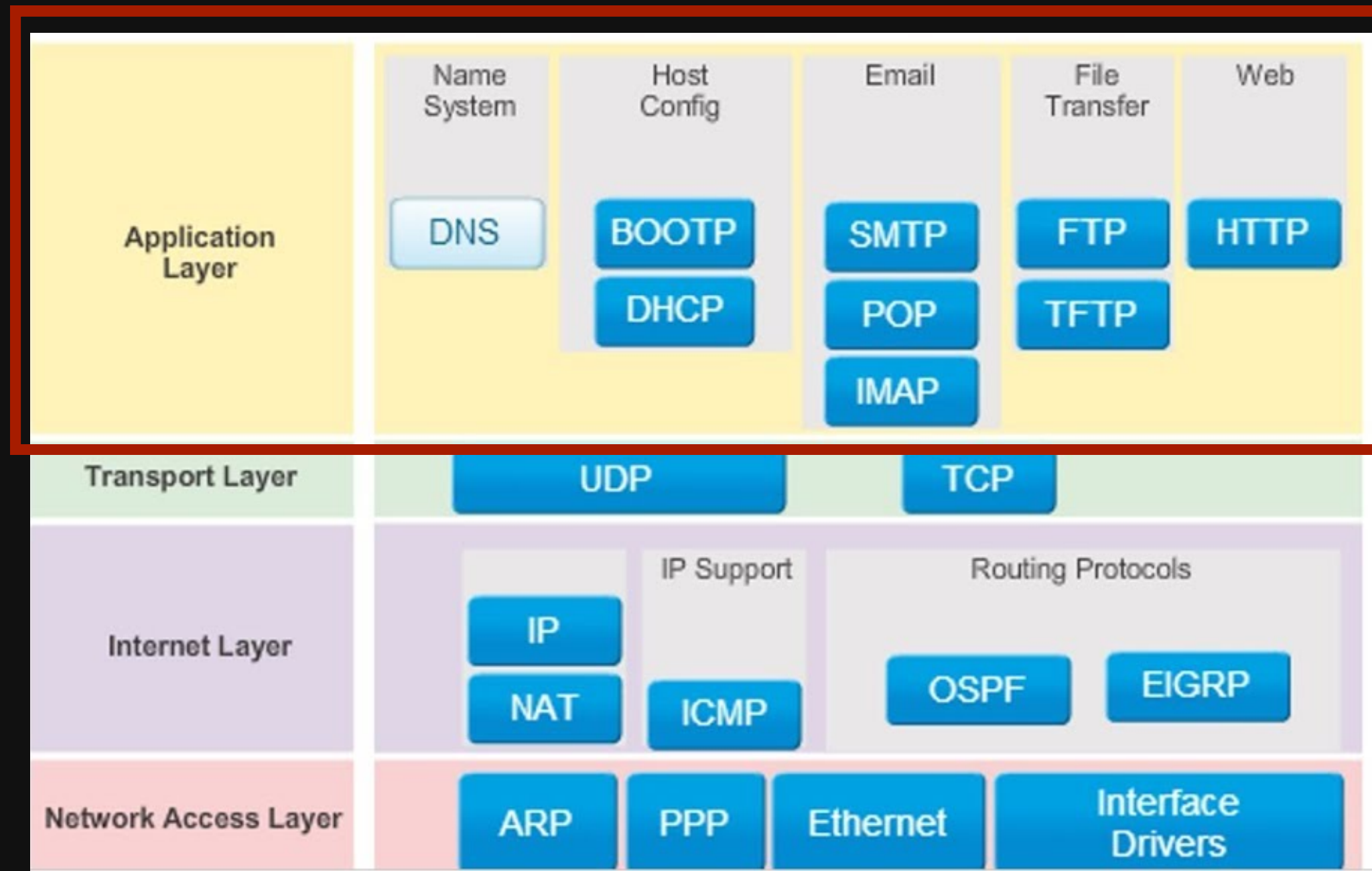
This is not a networking course:

- **Network:** A group of interconnected computers that can communicate
- **Internet:** A global infrastructure for networking computers around the entire world together
- **World Wide Web:** A system of documents and resources linked together, accessible via URLs

Network Protocols

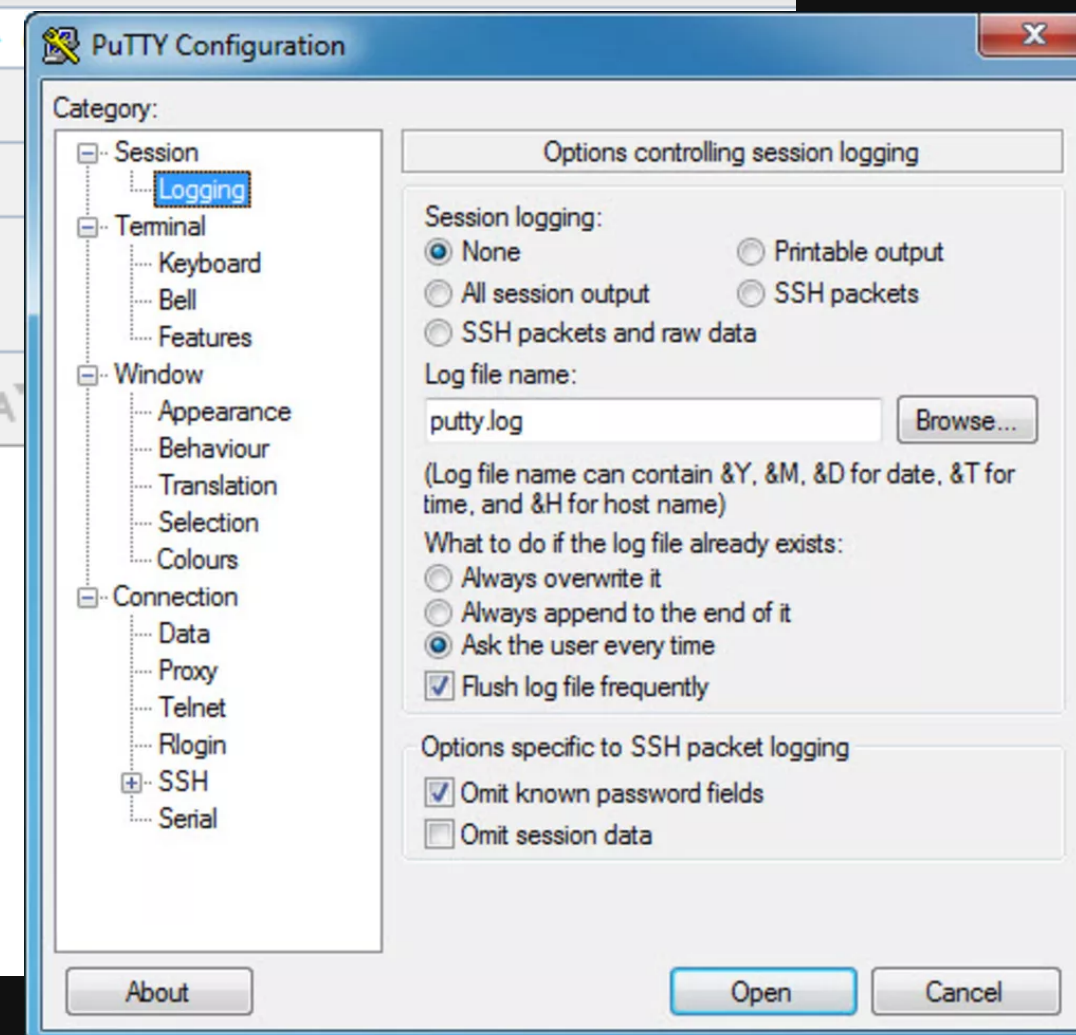
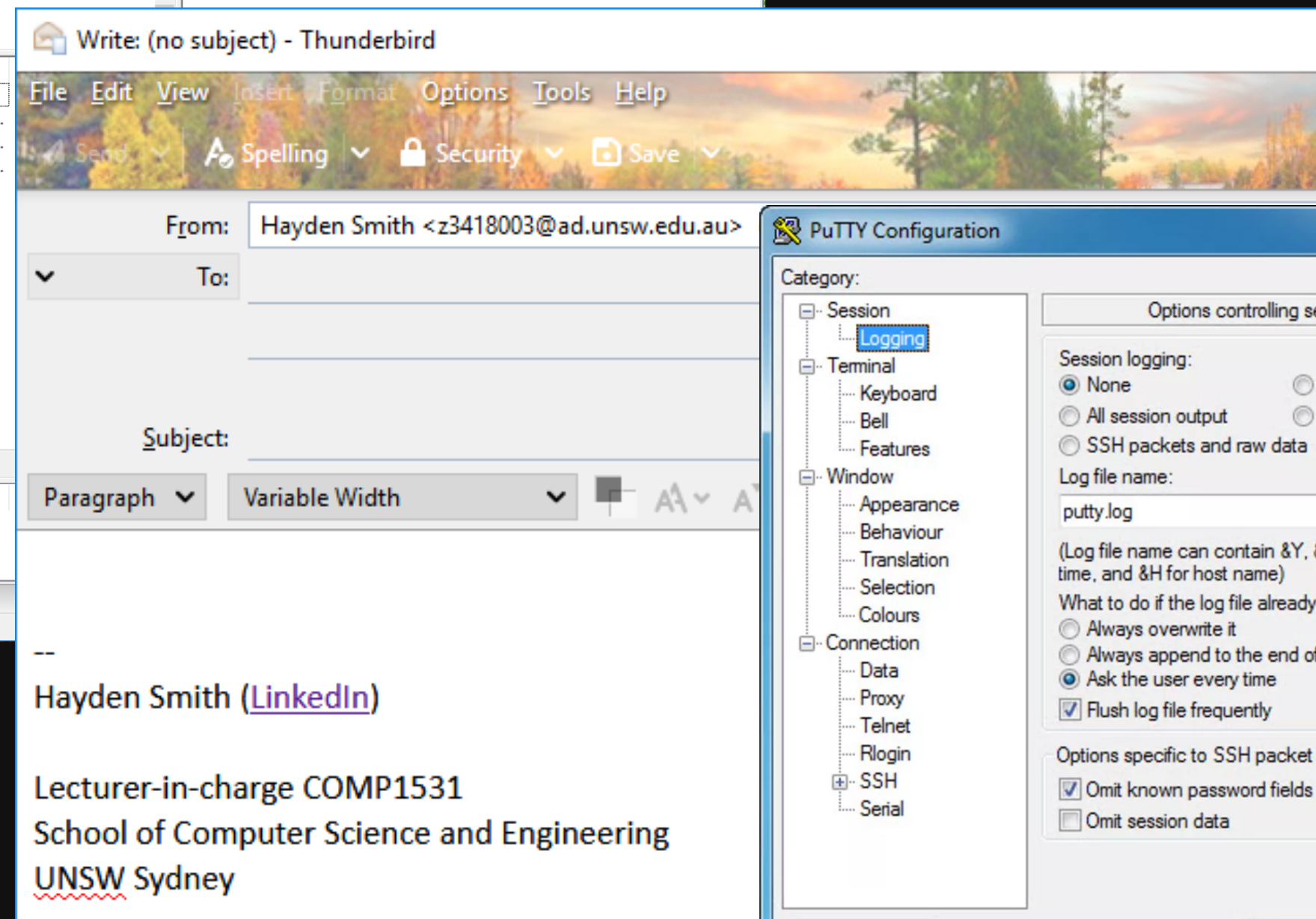
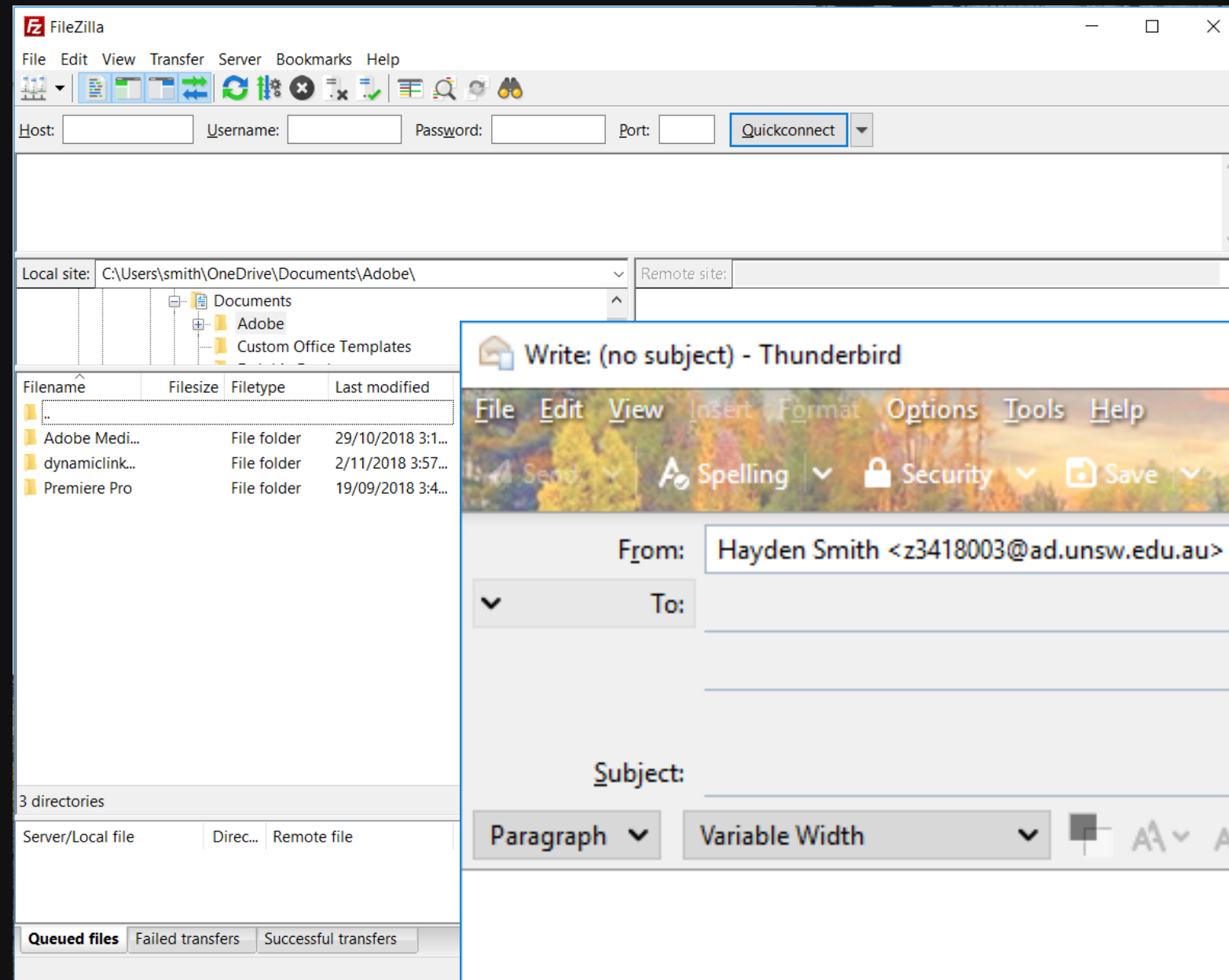
- Communication over networks must have a certain "structure" so everyone can understand
- Different "structures" (protocols) are used for different types of communication

Network Protocols



Source

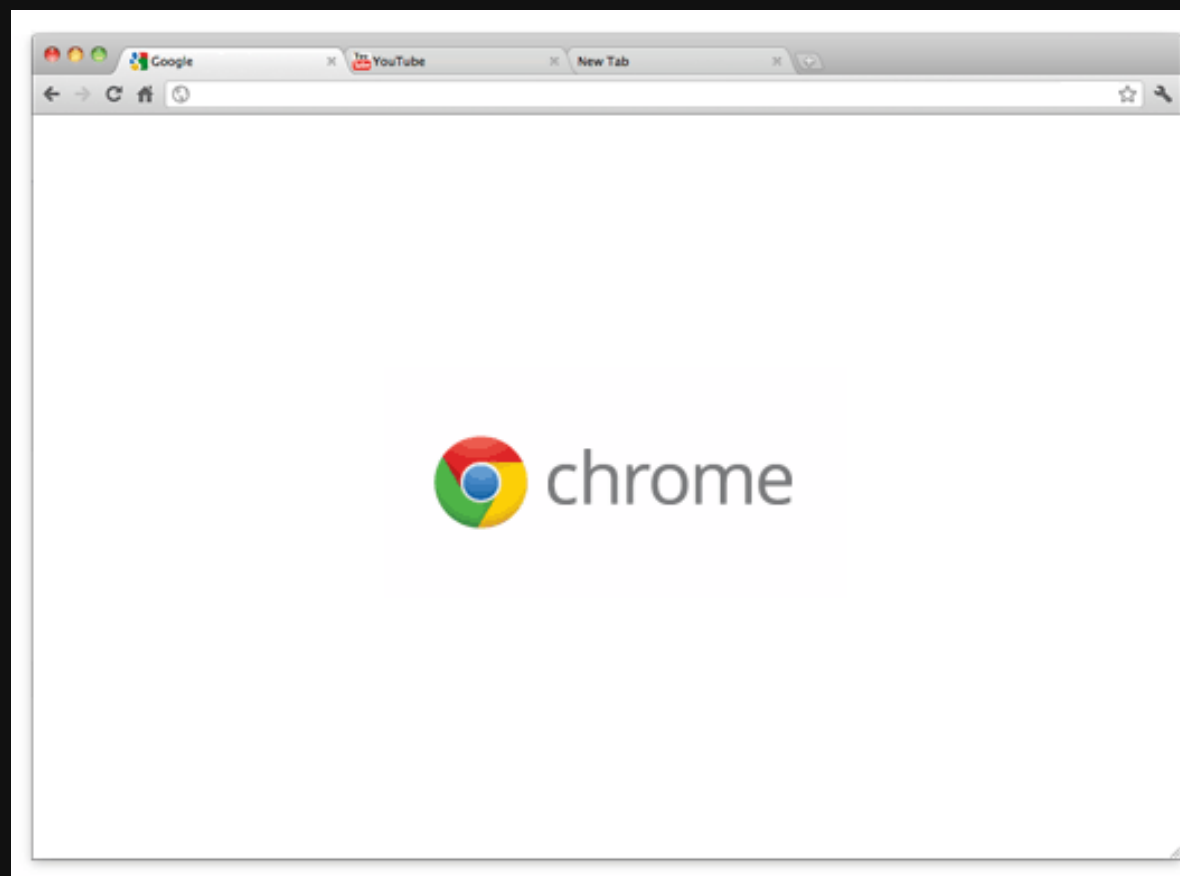
Examples?



HTTP

HTTP: Hypertext Transfer Protocol

I.E. Protocol for sending and receiving HTML documents (nowadays much more)



Request



Response



Web browsers are applications to request and receive HTTP

HTTP Request & Response

HTTP Request

```
1 GET /hello HTTP/1.1
2 Host: 127.0.0.1:5000
3 Connection: keep-alive
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.90 Safari/537.36
6 Sec-Fetch-Mode: navigate
7 Sec-Fetch-User: ?1
8 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
9 Sec-Fetch-Site: none
10 Accept-Encoding: gzip, deflate, br
11 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
```

HTTP Response

```
1 HTTP/1.0 200 OK
2 Content-Type: text/html; charset=utf-8
3 Content-Length: 12
4 Server: Werkzeug/0.16.0 Python/3.5.3
5 Date: Wed, 09 Oct 2019 13:21:51 GMT
6
7 Hello world!
```


Flask

Lightweight HTTP web server built in python

flask1.py

```
1 from flask import Flask
2 APP = Flask(__name__)
3
4 @APP.route("/")
5 def hello():
6     return "Hello World!"
7
8 if __name__ == "__main__":
9     APP.run()
```

```
1 $ python3 flask1.py
```

Server an image

Time to serve an image via a flask server...

flask2.py

```
1 from flask import Flask, send_file
2 APP = Flask(__name__)
3
4 @APP.route("/img")
5 def img():
6     return send_file('./cat.jpg', mimetype='image/jpeg')
7
8 if __name__ == "__main__":
9     APP.run()
```

```
1 $ python3 flask2.py
```

Flask Reloading

Lightweight HTTP web server built in python

flask1.py

```
1 from flask import Flask
2 APP = Flask(__name__)
3
4 @APP.route("/")
5 def hello():
6     return "Hello World!"
7
8 if __name__ == "__main__":
9     APP.run()
```

```
1 $ export FLASK_APP=flask1.py
2 $ export FLASK_DEBUG=1
3 $ python3 -m flask run
```

Learn More

Some tutorials include:

1. <https://pythonspot.com/flask-web-app-with-python/>
2. <https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask>

When it comes to online tutorials, note that:

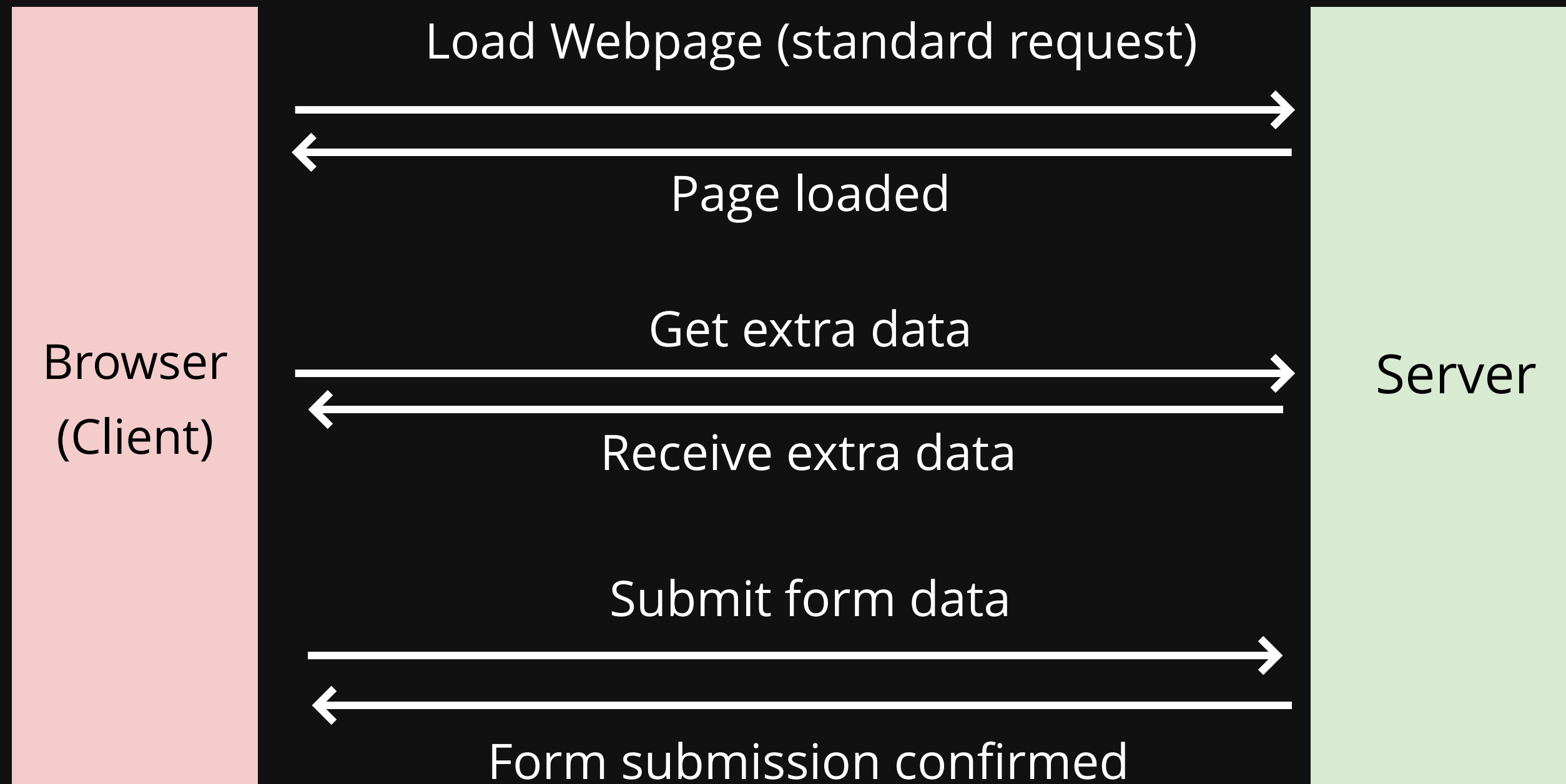
- Each "tutorial" may be using different python versions
- Each "tutorial" may have different aims in mind

API

An API (Application Programming Interface) refers to an interface exposed by a particular piece of software.

The most common usage of "API" is for Web APIs, which refer to a "contract" that a particular service provides. The interface of the service acts as a black box and indicates that for particular endpoints, and given particular input, the client can expect to receive particular output.

Web API



Restful API & "CRUD"

A *RESTful API* is an application program interface (*API*) that uses HTTP requests to GET, PUT, POST and DELETE data. These 4 methods describe the "nature" of different API requests.

GET, PUT, POST, DELETE are HTTP Methods

Method	Operation
POST	Create
GET	Read
PUT	Update
DELETE	Delete

Input & Output

Different CRUD properties require different approaches for input. All output are the same.

Inputs are either:

- GET: via URL and "request.args"
- PUT|POST|DELETE: via post-data and via "request.get_json()"
- All outputs should be packaged up as JSON
- (JSON discussed later)

crud.py

```
1 from flask import Flask, request
2 from json import dumps
3
4 APP = Flask(__name__)
5
6 @APP.route("/one", methods=['GET'])
7 def one():
8     return dumps({
9         '1': request.args.get('data1'),
10        '2': request.args.get('data1'),
11    })
12
13 @APP.route("/two", methods=['POST'])
14 def two():
15     data = request.get_json()
16     return dumps({
17         '1': data['data1'],
18         '2': data['data2'],
19    })
20
21 if __name__ == '__main__':
22     APP.run()
```


Using CRUD and state

Task:

Create a web server that uses CRUD to allow you to create, update, read, and delete a point via HTTP requests

Use a global variable to manage the state.

point.py

Talking to Flask

How can we talk to flask?

1. API client
2. Web Browser
3. URLLib via python

API Client (Postman)

How to download/install postman:

- Open google chrome
- Google "ARC client"
- Install the addon and open it
- Follow the demo in the lectures

You may need to use a tool like this in the final exam.

API Client (A R C)

ARC


Request

HTTP request

Socket

History


Send a request and recall it from here



Once you made a request it will appear in this place.

Saved

Save a request and recall it from here



Use ctrl+s to save a request. It will appear in this place.

Method
GET

Request URL

SEND

Parameters

Headers

Variables

Toggle source mode

Insert headers set

Header name

Header value

ADD HEADER

Headers are valid

Headers size: bytes

Install new ARC with new features!

Selected environment: Default

Web Browser

The screenshot shows a web browser window with a single tab titled '127.0.0.1:5000/hello'. The address bar also displays '127.0.0.1:5000/hello'. The main content area shows 'Hello World!'. The browser's developer tools are open, with the 'Network' tab selected. The network log shows a single request named 'hello'. The 'Headers' sub-tab is active, displaying the following information:

- General**
 - Request URL: `http://127.0.0.1:5000/hello`
 - Request Method: `GET`
 - Status Code: ● 200 OK
 - Remote Address: `127.0.0.1:5000`
 - Referrer Policy: `no-referrer-when-downgrade`
- Response Headers** [view source](#)
 - Content-Length: `12`
 - Content-Type: `text/html; charset=utf-8`
 - Date: `Wed, 09 Oct 2019 13:26:05 GMT`
 - Server: `Werkzeug/0.16.0 Python/3.5.3`
- Request Headers** [view source](#)
 - Accept: `text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3`
 - Accept-Encoding: `gzip, deflate, br`
 - Accept-Language: `en-GB,en-US;q=0.9,en;q=0.8`
 - Cache-Control: `max-age=0`

At the bottom of the network panel, it indicates '1 requests' and '166 B transferred'.

requests - Python

requests is a python3 library that allows you to programmatically make HTTP requests to a web server.

You will use this extensively in iteration 2.

requests - Python

echo.py

```
1 from flask import Flask, request
2 from json import dumps
3
4 APP = Flask(__name__)
5
6 @APP.route("/echo", methods=['GET'])
7 def echo():
8     return dumps({'data': request.args.get('data')})
9
10 if __name__ == '__main__':
11     APP.run()
```

echo_main.py

```
1 import json
2 import requests
3
4 def get_payload():
5     response = requests.get('http://127.0.0.1:5000/echo', params={'data': 'hi'})
6     payload = response.json()
7     print(payload)
8
9 if __name__ == '__main__':
10     get_payload()
```

We expect you to do your own research for POST

Web server as a wrapper

Because you've written so many **integration** tests for iteration 1, it makes sense to:

1. Implement all of the functions from iteration one
2. *Then* wrap them in a flask server

Web server as a wrapper

iter2example/search.py

```
1 def search(token, query_str):
2     return {
3         'messages' : [
4             'Hello ' + token + ' ' + query_str,
5             # Not the right structure
6         ]
7     }
```

iter2example/server.py

```
1 from json import dumps
2 from flask import Flask, request
3
4 from search import search_fn
5
6 APP = Flask(__name__)
7
8 @APP.route('/search', methods=['GET'])
9 def search_flask():
10     return dumps(search(request.args.get('token'), request.args.get('query_str')))
11
12 if __name__ == '__main__':
13     APP.run()
```

(Bonus) interesting question

How do companies track whether or
not you've read an email they've sent
you?