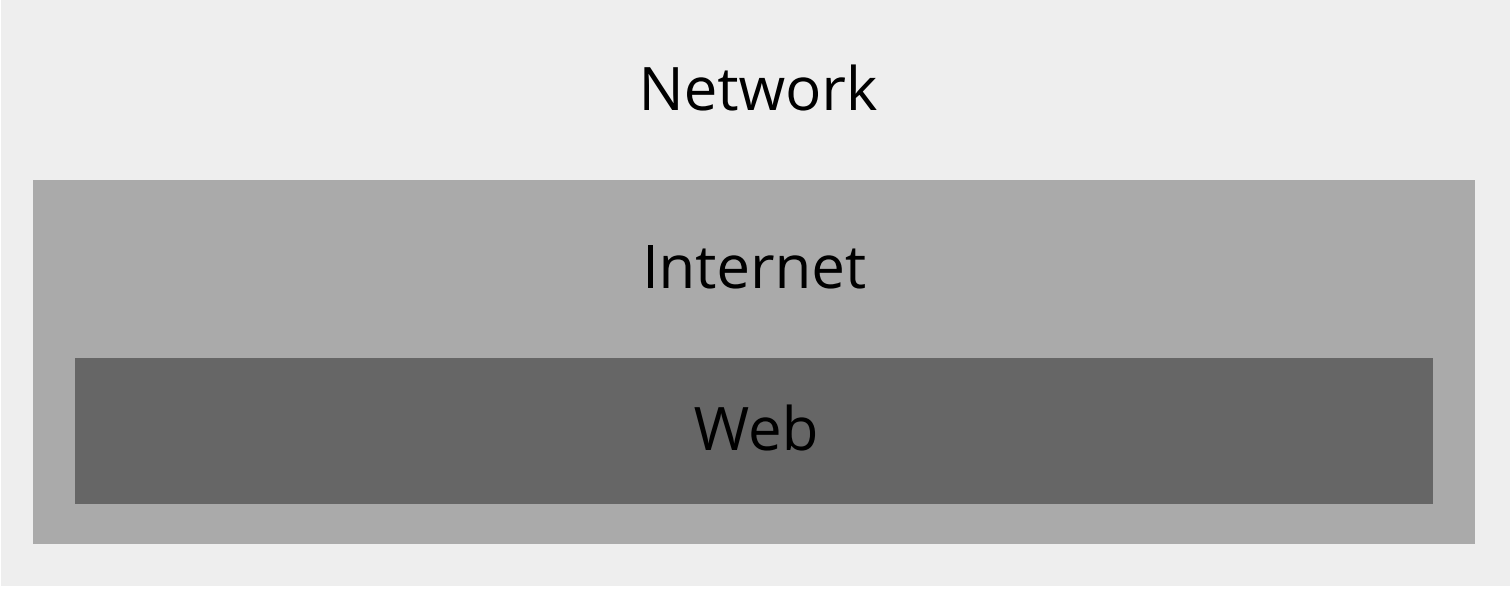
COMP1531

4.1 - HTTP, Flask

Computer Networks



The network

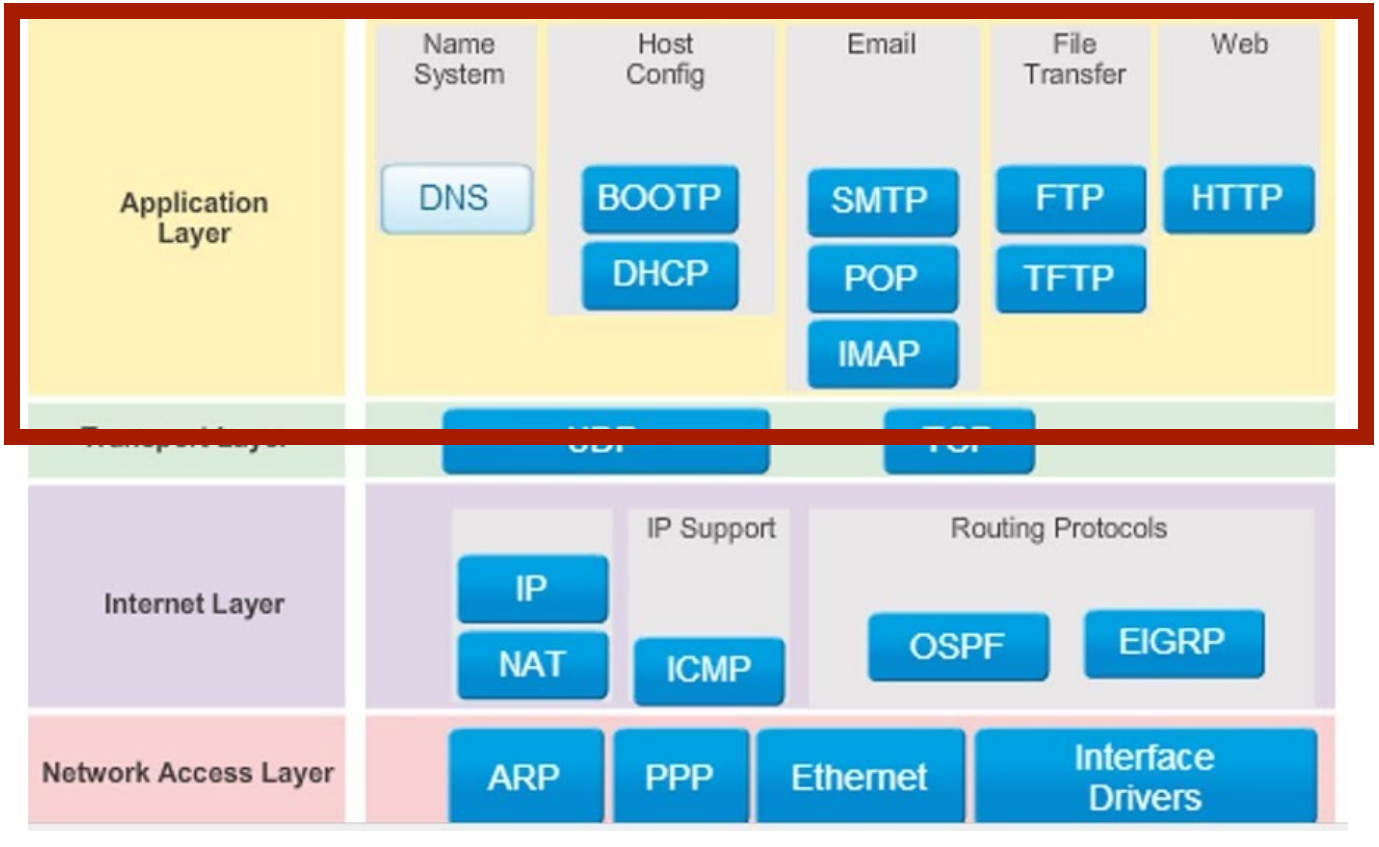
This is not a networking course:

- **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

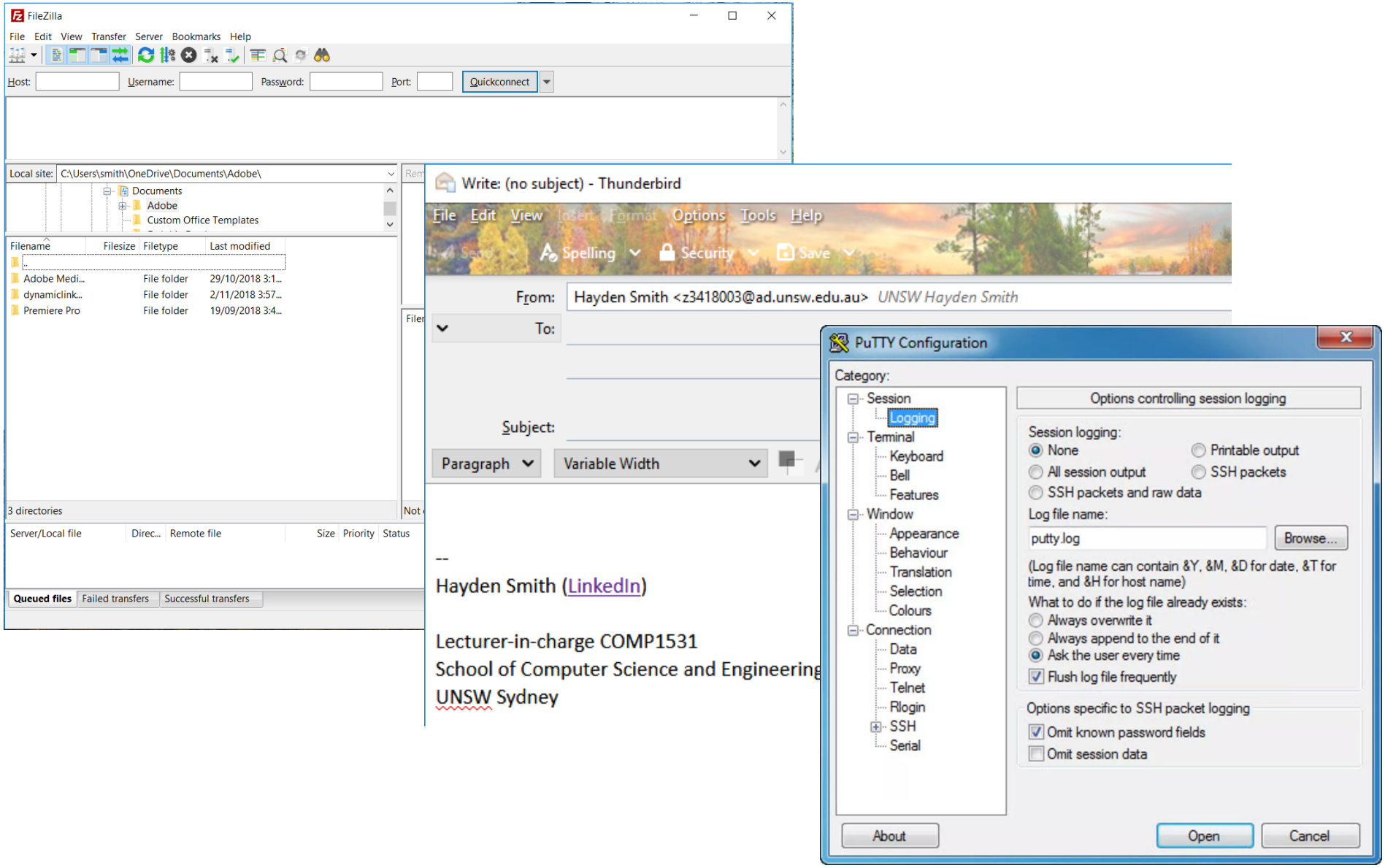
- Communication over networks must have a certain "structure" so everyone can understand
- Different "structures" (protocols) are used for different types of communication

Network Protocols

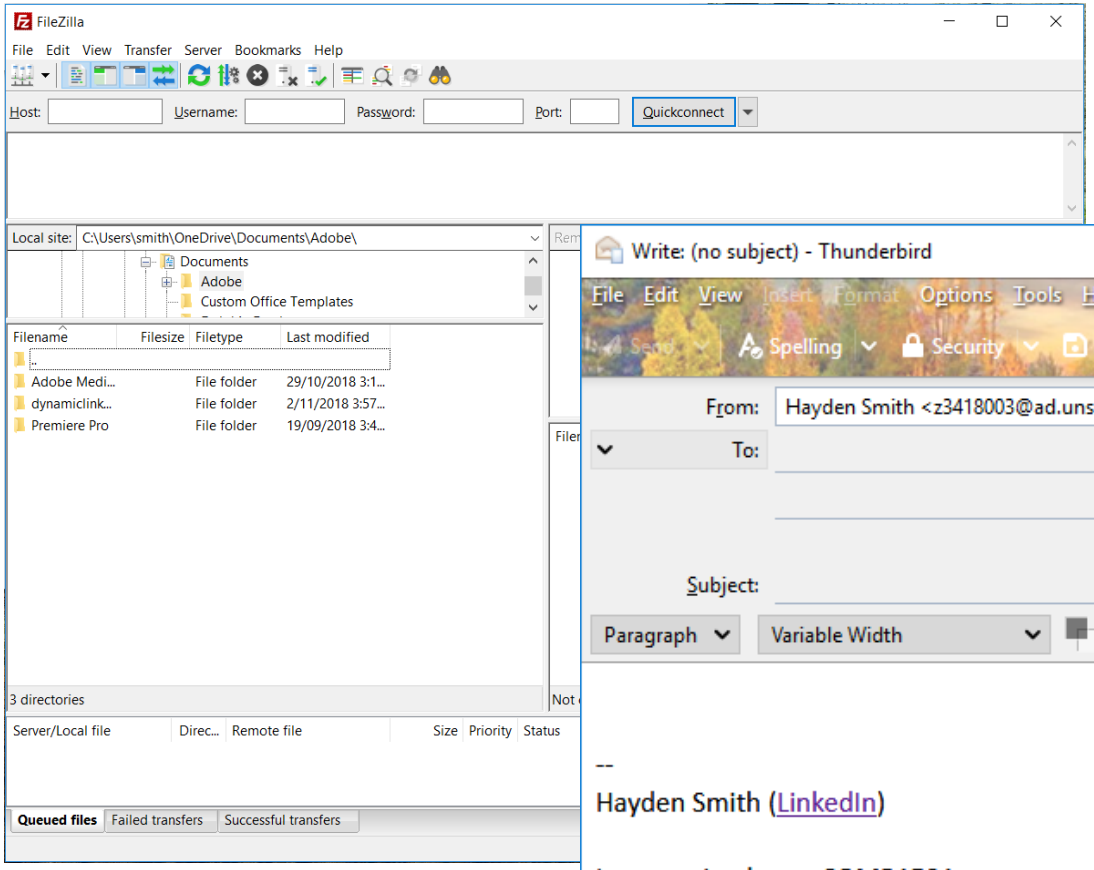


Source

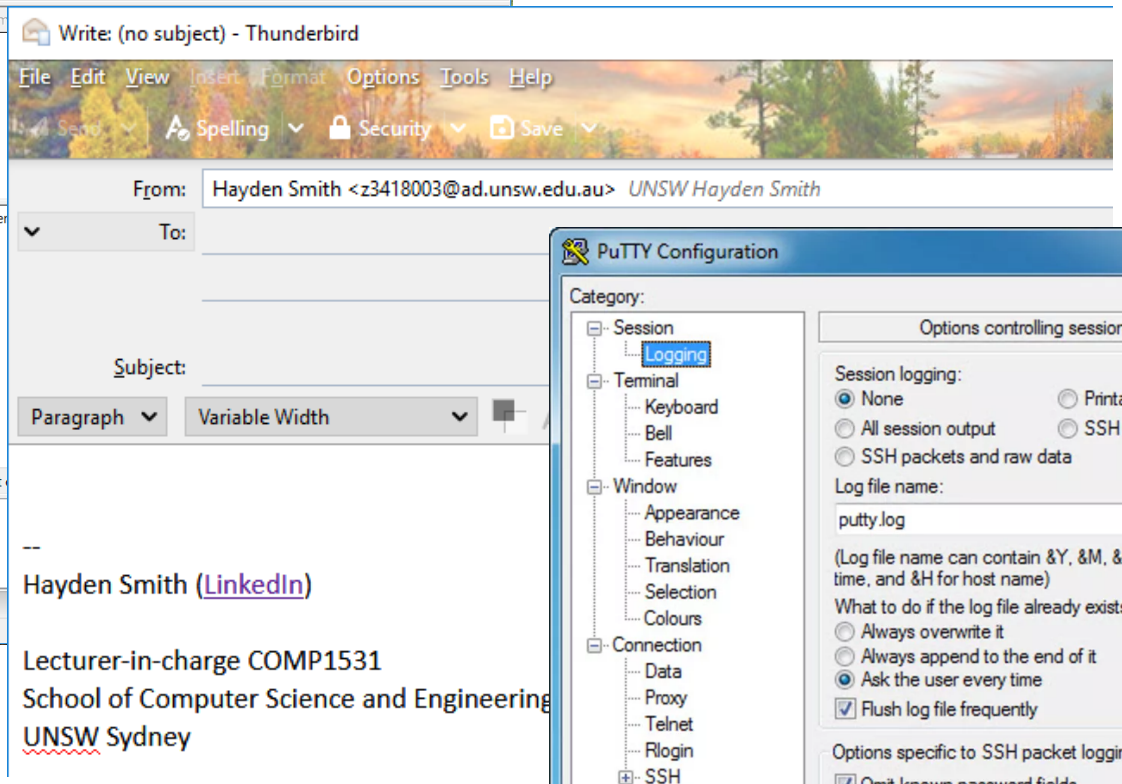
Examples?


FileZilla
File Edit View Transfer Server Bookmarks Help
Host: Username: Password: Port: Quickconnect
Local site: C:\Users\smith\OneDrive\Documents\Adobe\
Documents
Adobe
Custom Office Templates
Filename Filesize Filetype Last modified
Adobe Medi... File folder 29/10/2018 3:1...
dynamiclink... File folder 2/11/2018 3:57...
Premiere Pro File folder 19/09/2018 3:4...
3 directories
Server/Local file Direc... Remote file Size Priority Status
Queued files Failed transfers Successful transfers
Write: (no subject) - Thunderbird
File Edit View Insert Format Options Tools Help
Send Spelling Security Save
From: Hayden Smith <z3418003@ad.unsw.edu.au> UNSW Hayden Smith
To:
Subject:
Paragraph Variable Width
--
Hayden Smith (LinkedIn)
Lecturer-in-charge COMP1531
School of Computer Science and Engineering
UNSW Sydney
PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Options controlling session logging
Session logging:
None
Printable output
All session output
SSH packets
SSH packets and raw data
Log file name:
putty.log
Browse...
(Log file name can contain &Y, &M, &D for date, &T for time, and &H for host name)
What to do if the log file already exists:
Always overwrite it
Always append to the end of it
Ask the user every time
Flush log file frequently
Options specific to SSH packet logging
Omit known password fields
Omit session data
About
Open
Cancel

Examples?



FTP



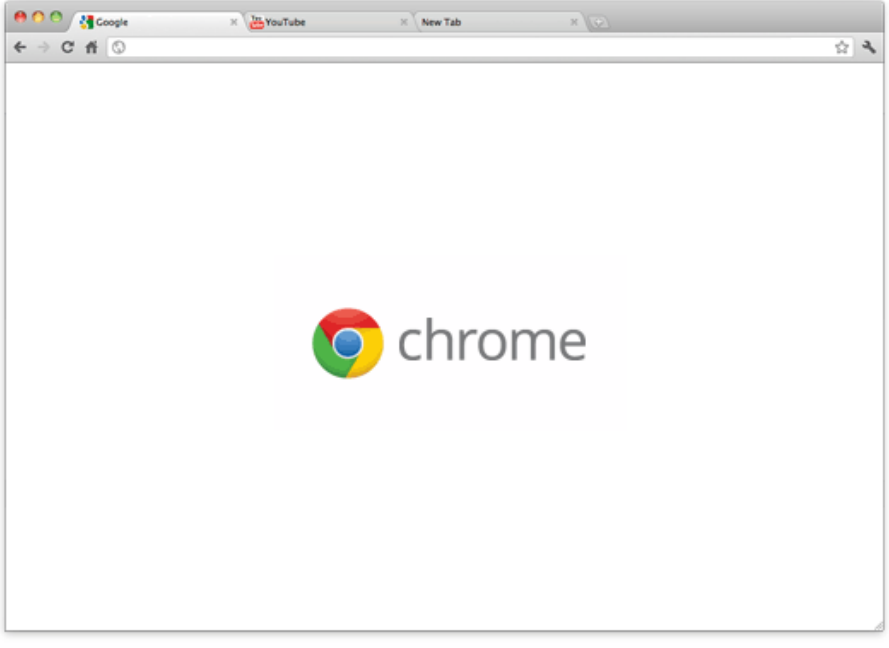
IMAP

SSH

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

```
GET /hello HTTP/1.1
Host: 127.0.0.1:5000
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Ge
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;
Sec-Fetch-Site: none
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
```

HTTP Response

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 12
Server: Werkzeug/0.16.0 Python/3.5.3
Date: Wed, 09 Oct 2019 13:21:51 GMT

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/jpg')
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 $ FLASK_APP=flask1.py
2 $ FLASK_DEBUG=1
3 $ 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

Browser (Client)

Load Webpage (standard request)

Page loaded

Get extra data

Receive extra data

Submit form data

Form submission confirmed

Server

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	**C**reate
GET	**R**ead
PUT	**U**pdate
DELETE	**D**elete

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)

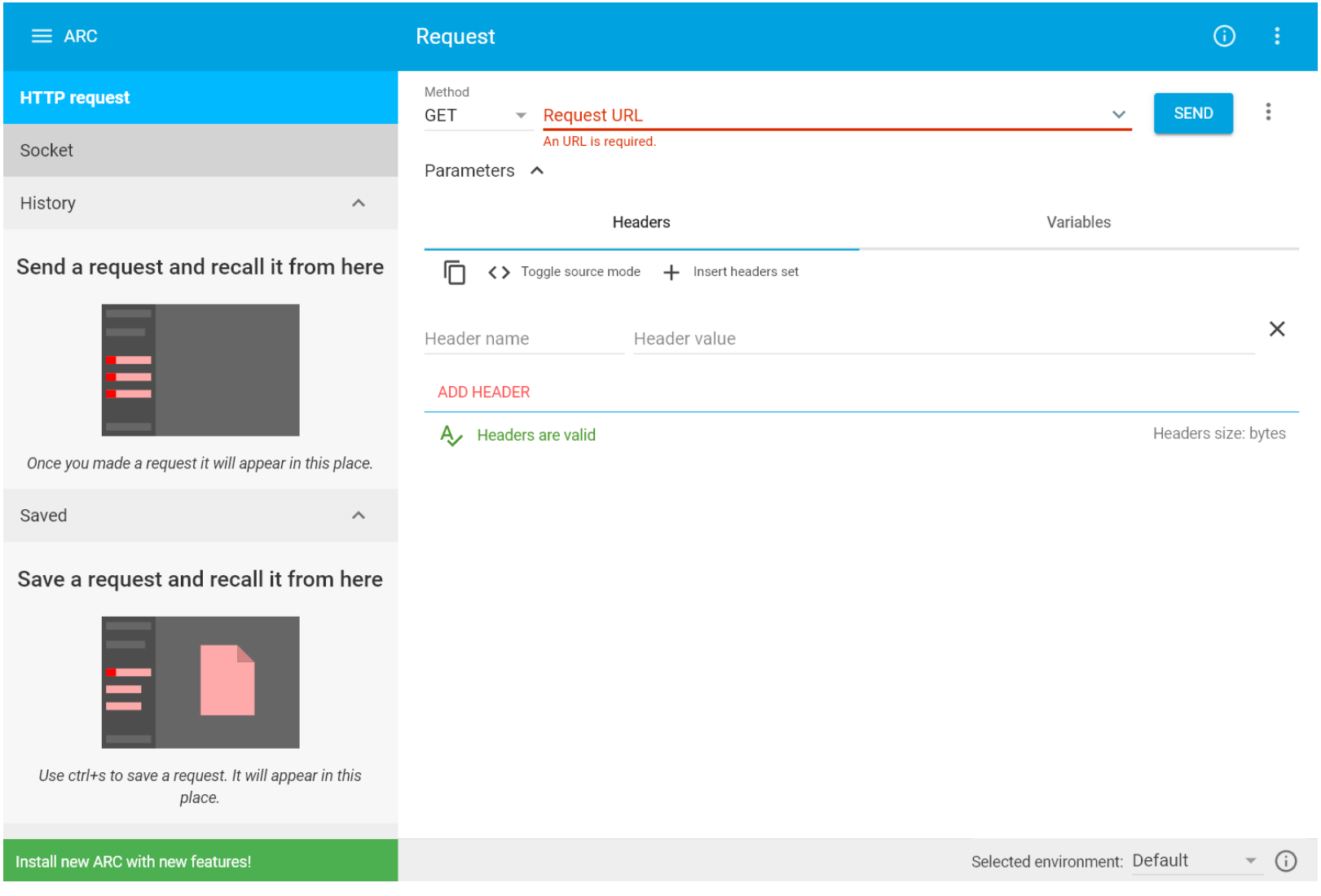
How to download/install postman:

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

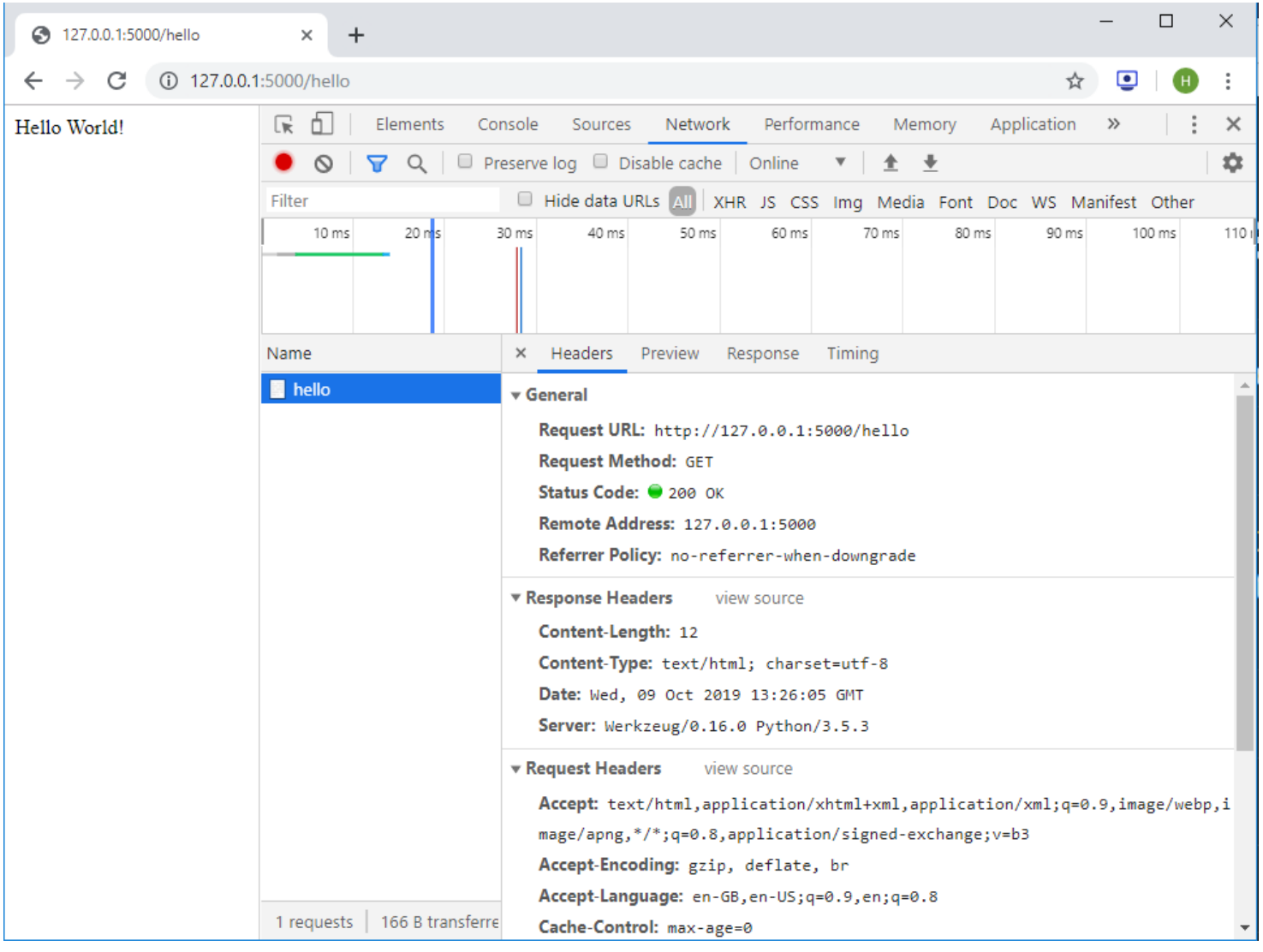
Other notes:

- There are many of these types of apps
- Think of it like "a GUI for cURL"
- You may be expected to use postman (or equivalent) in a final exam

API Client (A R C)



Web Browser


127.0.0.1:5000/hello
127.0.0.1:5000/hello
Hello World!
Elements
Console
Sources
Network
Performance
Memory
Application
Preserve log
Disable cache
Online
Filter
Hide data URLs
All
XHR
JS
CSS
Img
Media
Font
Doc
WS
Manifest
Other
10 ms
20 ms
30 ms
40 ms
50 ms
60 ms
70 ms
80 ms
90 ms
100 ms
Name
hello
Headers
Preview
Response
Timing
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
1 requests
166 B transferre

urllib - Python

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

You will use this extensively in iteration 2.

urllib - Python

echo.py

```python
from flask import Flask, request
from json import dumps

APP = Flask(__name__)

@APP.route("/echo", methods=['GET'])
def echo():
    return dumps({'data': request.args.get('data')})

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

echo_main.py

```python
import json
import urllib.request

def get_payload():
    response = urllib.request.urlopen('http://127.0.0.1:5000/echo?data=hi')
    payload = json.loads(response.read().decode('utf8'))
    print(payload)

if __name__ == '__main__':
    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(requests.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?