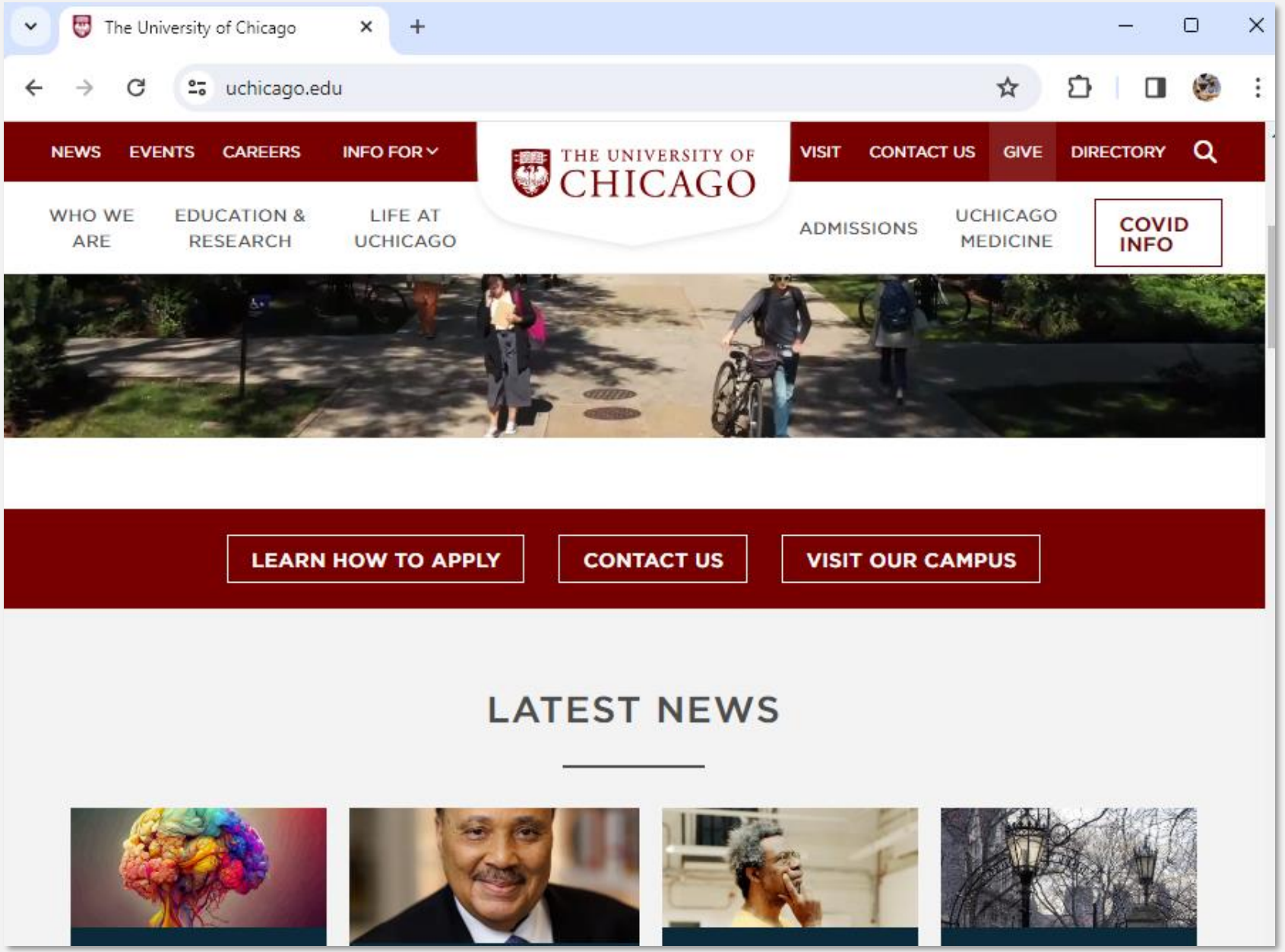
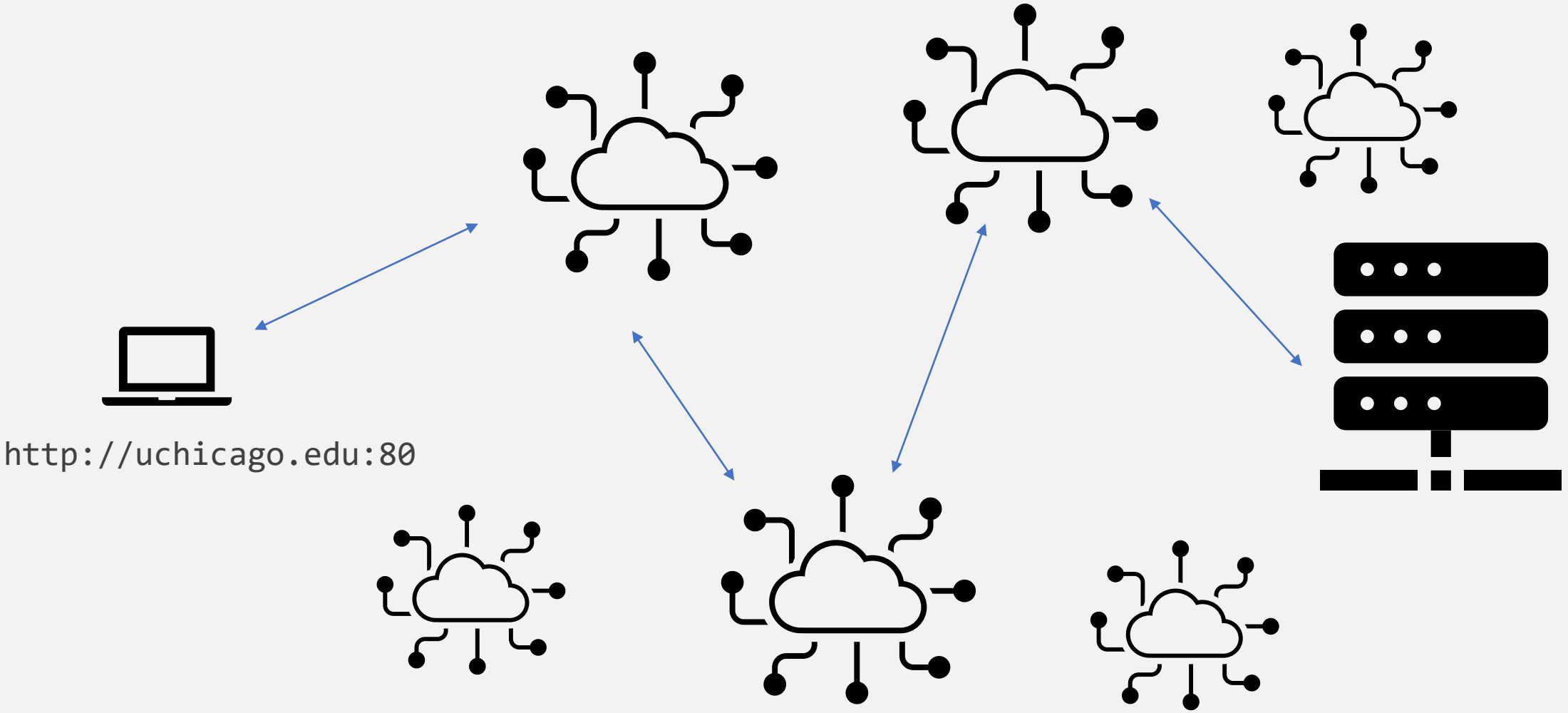
The web, under the hood

By Shahbaz Chaudhary

What happens when you visit a website?

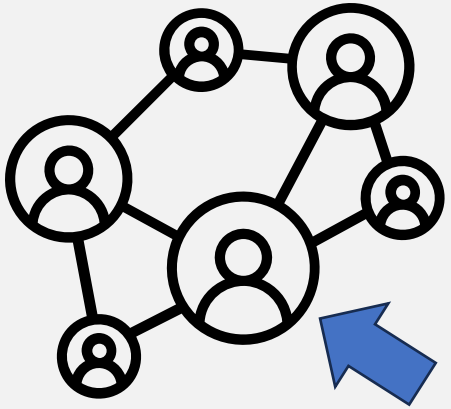


The browser appends a port number: 80



http://uchicago.edu:80

Detour: ports and IP addresses (simplified)



68.45.56.20

Each computer on a network has a unique IP address. The internet is made up of many networks. Get your current address by using the `ifconfig` (or `ipconfig`) command.

A machine may be running many servers: web server, time server, ssh server, etc.

How do we differentiate among services?
Using ports!

http (web): port 80
ssh: port 22
ntp (etwork time protocol): port 123

```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : hsd1.in.comcast.net.
   IPv6 Address. . . . . . . . . . . : 2601:801:200:7ec0::2
   IPv6 Address. . . . . . . . . . . : 2601:801:200:7ec0:39d9:a7b9:d3e6:3fd4
   Temporary IPv6 Address. . . . . . : 2601:801:200:7ec0:6519:3a44:524c:2e25
   Link-local IPv6 Address . . . . . : fe80::26e4:8d4e:e3d7:69c%21
   IPv4 Address. . . . . . . . . . . : 192.168.0.4
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.0.1
```

Modern internet uses packets while old phone network was a *very* long cable

Notice that data going from your computer to a remote server can take different paths (`tracert` or `traceroute`)

The ability to take different routes means the network is route around damaged nodes.

Makes sense, since the original network was designed to survive nuclear war!

Pro-tip: check if a machine is down by doing `ping machine_ip`

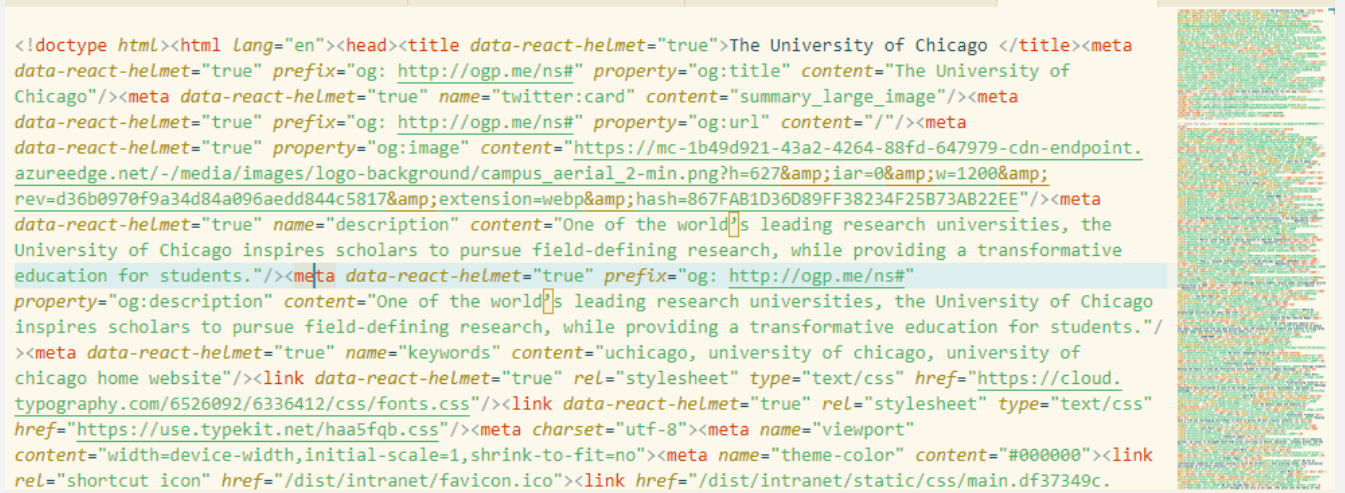
```
C:\Users\shahb>tracert cnn.com

Tracing route to cnn.com [151.101.131.5]
over a maximum of 30 hops:

  1     3 ms     5 ms     5 ms  192.168.0.1
  2    13 ms    10 ms    11 ms  96.120.112.149
  3    14 ms    11 ms    20 ms  po-303-1215-rur01.bloomington.in.indiana.comcast.net [96.110.168.21]
  4    10 ms    12 ms    11 ms  po-2-rur02.bloomington.in.indiana.comcast.net [96.108.120.86]
  5    16 ms    17 ms    19 ms  be-50-rar01.kokomo.in.indiana.comcast.net [96.108.120.145]
  6    24 ms    22 ms    22 ms  be-5-ar01.area4.il.chicago.comcast.net [24.153.88.85]
  7    24 ms    23 ms    23 ms  69.241.116.146
  8    22 ms    22 ms    23 ms  151.101.131.5

Trace complete.
```

```
C:\Users\shahb>tracert cnn.com

Tracing route to cnn.com [151.101.67.5]
over a maximum of 30 hops:

  1     2 ms     2 ms     1 ms  192.168.0.1
  2    11 ms    13 ms    10 ms  96.120.112.149
  3     9 ms    11 ms     9 ms  po-303-1215-rur01.bloomington.in.indiana.comcast.net [96.110.168.21]
  4    11 ms    11 ms    15 ms  po-2-rur02.bloomington.in.indiana.comcast.net [96.108.120.86]
  5    15 ms    15 ms    15 ms  be-50-rar01.kokomo.in.indiana.comcast.net [96.108.120.145]
  6    28 ms    22 ms    22 ms  be-5-ar01.area4.il.chicago.comcast.net [24.153.88.85]
  7    21 ms    23 ms    24 ms  69.241.116.130
  8    23 ms    24 ms    24 ms  151.101.67.5

Trace complete.
```

HTML, CSS and JavaScript

What exactly is a web page?

Try “View page source” or open developer tools to get a peek at the code



HTML (Hyper text *markup language*) contains the content of a web page

```
<html>

 <head>
 <link rel="stylesheet" href="css/layout.css" type="text/css">
 <script type="text/javascript" src="script.js"></script>
 </head>

 <body>
  <p> This is a paragraph. It can contain a block of text and can
  contain multiple lines </p> HTML can contain <a href="http://uchicago.edu"> links </a>.
 </body>

</html>
```

HTML is a markup language, not a full language like Python or Javascript. (This is not a value judgement)

CSS (Cascading Style Sheets) controls the look & feel (and behavior) of HTML

Web developers often let the designers work with CSS. Non-designers, such as myself, often use CSS frameworks, such as Bootstrap or Tailwind.

See developer tools for a full list of CSS properties

```css
/* https://www.w3schools.com/css/ */

body {
 background-color: lightblue;
}

h1 {
 color: white;
 text-align: center;
}

p {
 font-family: verdana;
 font-size: 20px;
}
```

Javascript is the official programming language of the web (and the only one*)

```
//https://www.w3schools.com/jsref/event_onclick.asp

<h3 id="demo" onclick="myFunction()">Click me to change my color.</h3>

<script>
function myFunction() {
 document.getElementById("demo").style.color = "red";
}
</script>
```

* Newer technologies such as WASM will allow more and more languages to be available

There are convenient online tools to test http + css + js

https://jsfiddle.net/

https://jsbin.com/?html,output

Web pages can be dynamically generated

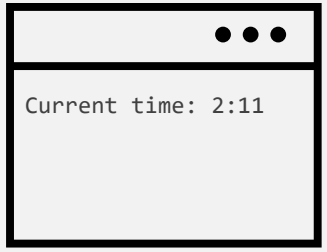
Python
(server side)

```python
from datetime import datetime
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
  return f"<p>Current time:{datetime.now().strftime("%H:%M")}</p>"
```

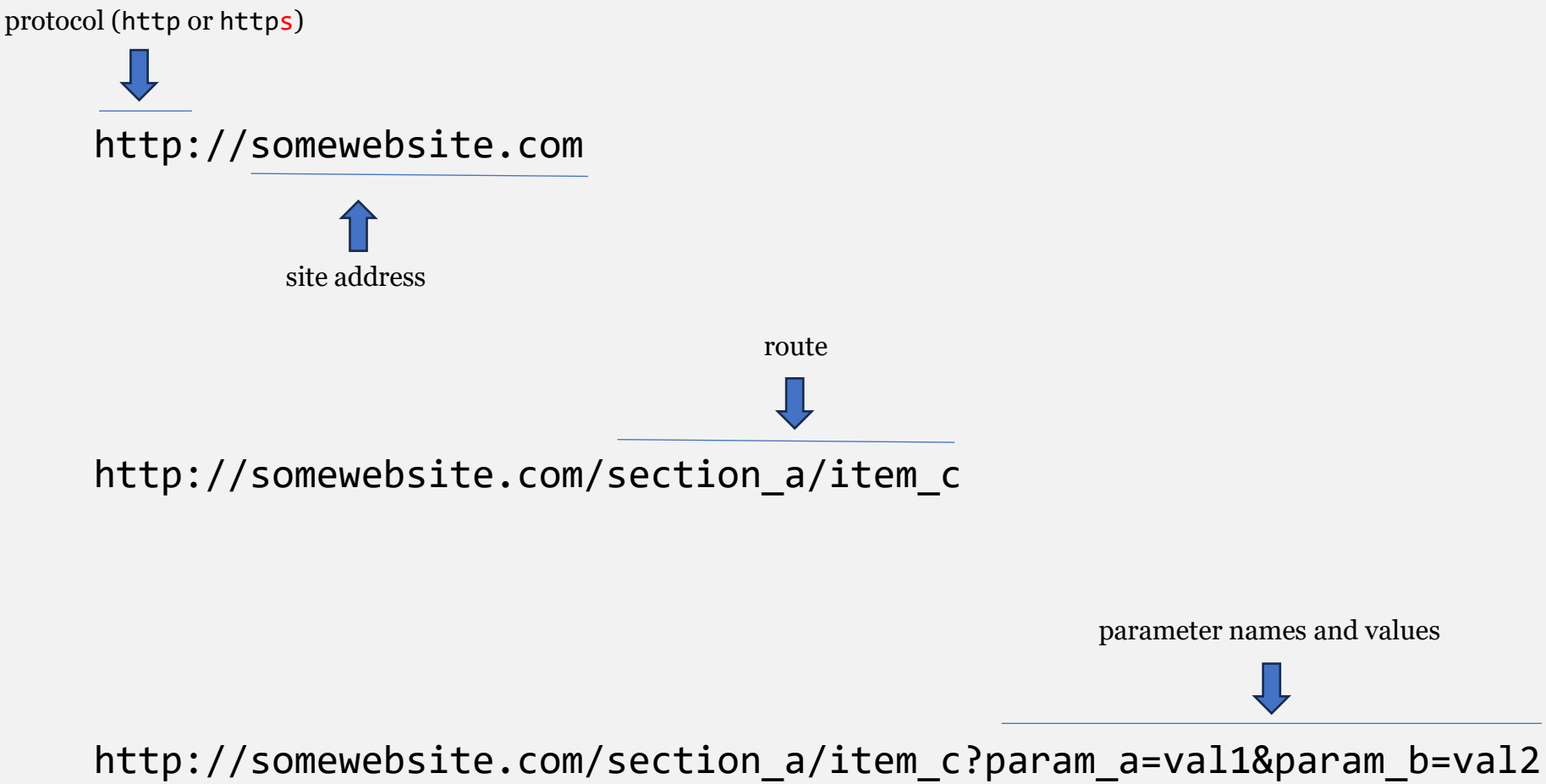
Generated HTML
(server side)

```html
<p>Current time: 2:11</p>
```

Browser view
(client side)



Web URLs contain lots of information



Hands on keyboard ...

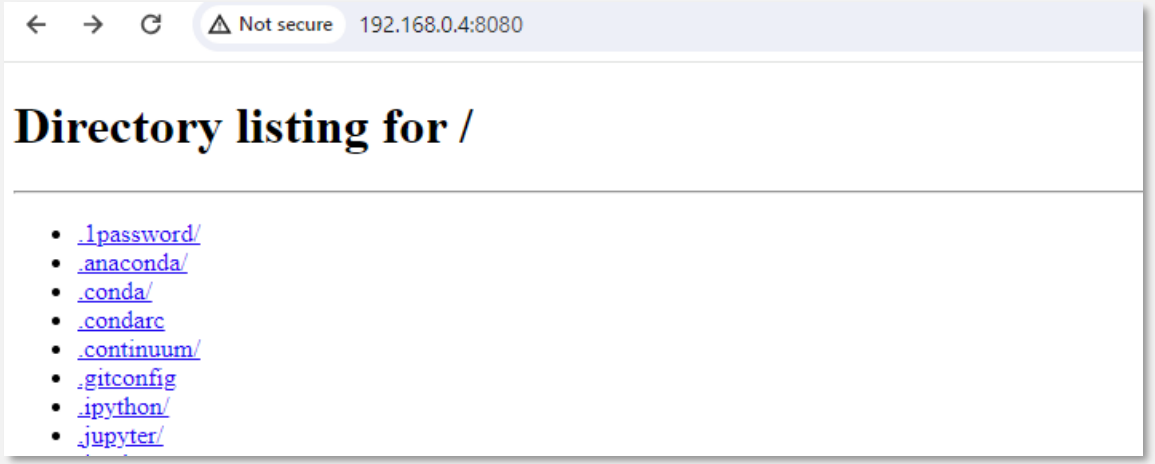
Run your own web server
python –m http.server 8080

What url can you type in your browser to view your local “web site?”

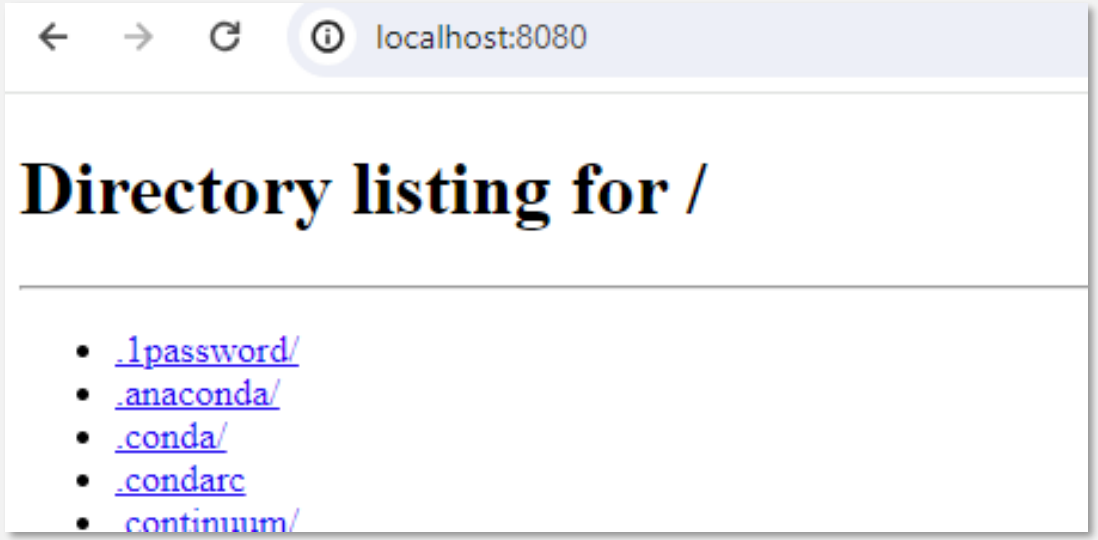
Get the assigned IP using `ipconfig` or `ifconfig`

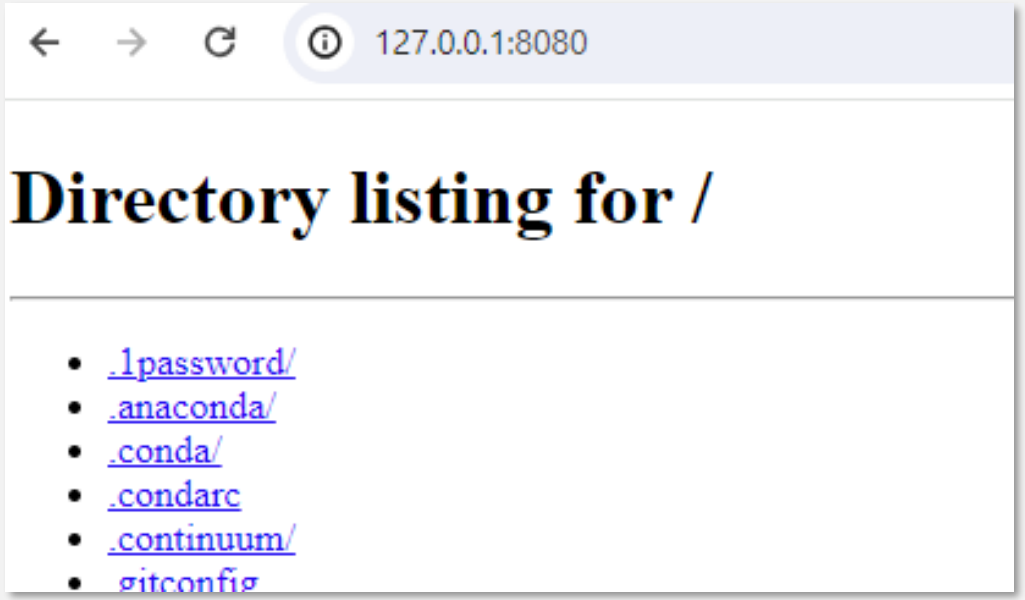
```
Connection-specific DNS Suffix  . : hsd1.in.comcast.net.
IPv6 Address. . . . . . . . . . . : 2601:801:200:7ec0::2
IPv6 Address. . . . . . . . . . . : 2601:801:200:7ec0:39d9:a7b9:d3e6:3fd4
Temporary IPv6 Address. . . . . . : 2601:801:200:7ec0:6519:3a44:524c:2e25
Link-local IPv6 Address . . . . . : fe80::26e4:8d4e:e3d7:69c%21
IPv4 Address. . . . . . . . . . . : 192.168.0.4
Subnet Mask . . . . . . . . . . . : 255.255.255.0
Default Gateway . . . . . . . . . : 192.168.0.1
```

Visit that url in the browser (remember to add the port)



Just use special names: `127.0.0.1` or `localhost`

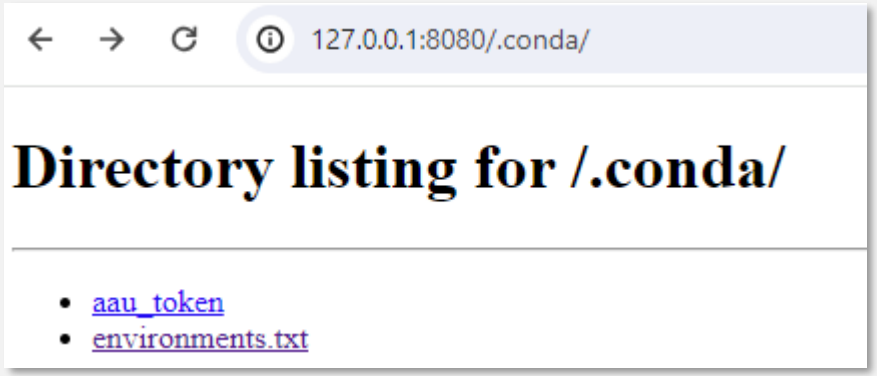


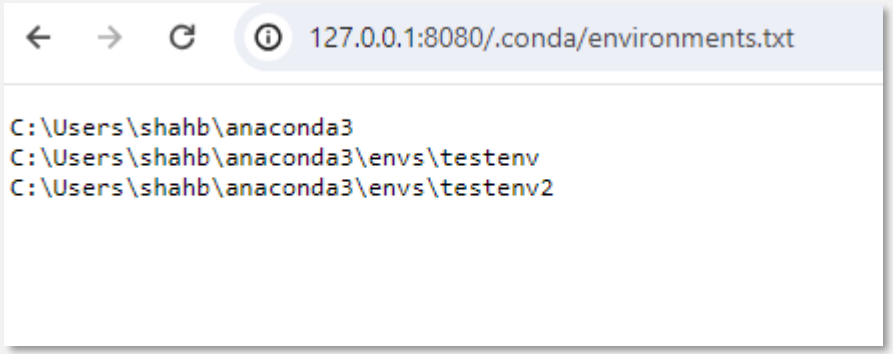


Look at the server logs

Each time the browser asks for more information, the server displays the log for it

The server prints the
- IP address]
- the date and time
- “GET”
- the route
- “HTTP/1.1”
- 200






```
(base) C:\Users\shahb>python -m http.server 8080
Serving HTTP on :: port 8080 (http://[::]:8080/) ...
::1 - - [17/Jan/2024 15:25:23] "GET / HTTP/1.1" 200 -
::ffff:127.0.0.1 - - [17/Jan/2024 15:25:43] "GET / HTTP/1.1" 200 -
::ffff:127.0.0.1 - - [17/Jan/2024 15:27:24] "GET /.conda/ HTTP/1.1" 200 -
::ffff:127.0.0.1 - - [17/Jan/2024 15:27:29] "GET /.conda/environments.txt HTTP/1.1" 200 -
```

HTTP protocol

Web log contents

```
::ffff:127.0.0.1 - - [17/Jan/2024 15:25:43] "GET / HTTP/1.1" 200 -
::ffff:127.0.0.1 - - [17/Jan/2024 15:27:24] "GET /.conda/ HTTP/1.1" 200 -
::ffff:127.0.0.1 - - [17/Jan/2024 15:27:29] "GET /.conda/environments.txt HTTP/1.1" 200 -
```

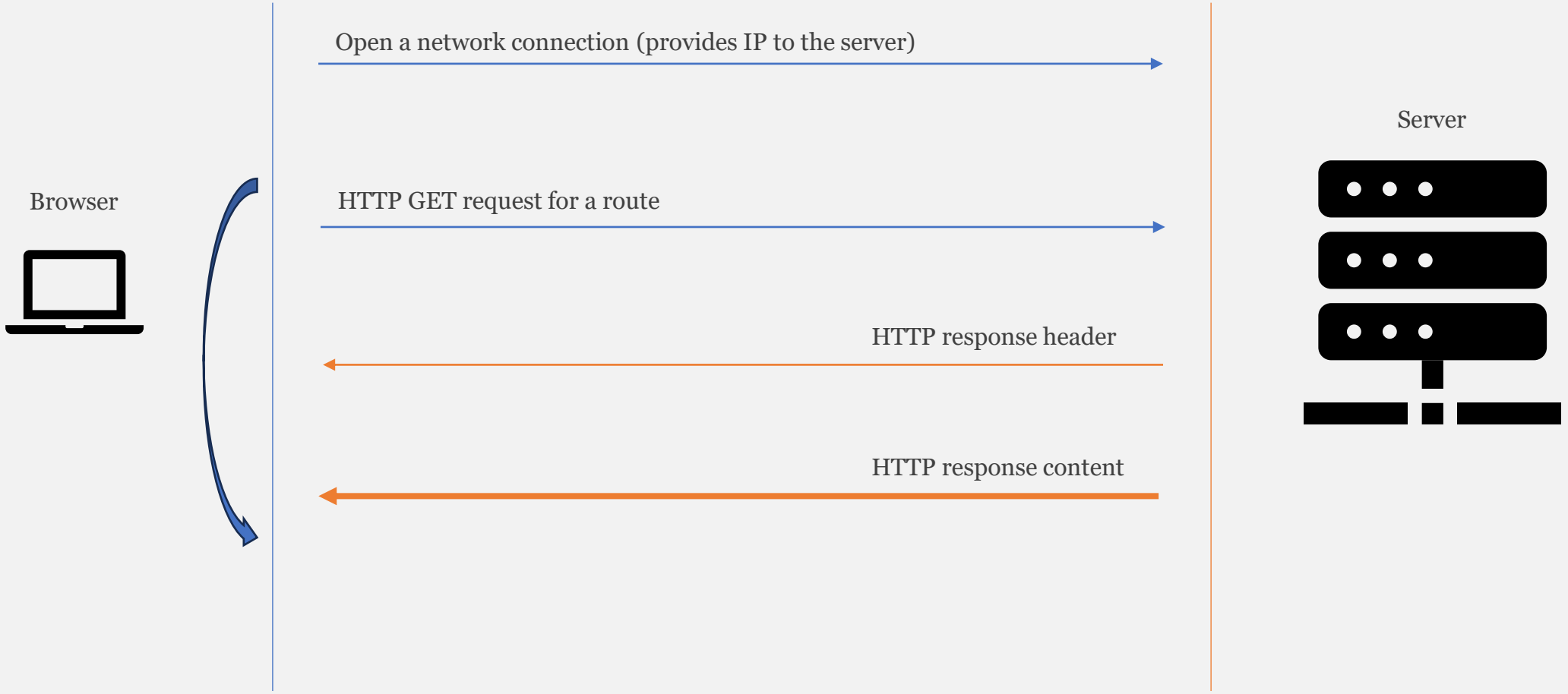
`GET`
The HTTP protocol defines some methods, such as GET, POST, PUT, DELETE. GET translates to “read a resources at the given route”

`/.conda/environments.txt`
This is the route and the resource requested by the browser

`HTTP/1.1`
The HTTP protocol version specifics generally are negotiated by the browser and the server and are of little concern to developers

`200`
Each request from the browser results in a response from the server, along with a response code. 200 means everything is ok. Do you know the code for a resource which isn’t found? 404!

What is exchanged between your browser and the server?



What is exchanged between your browser and the server?

1. Browser request

```
GET /.conda/ HTTP/1.1
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
bp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cache-Control: max-age=0
Connection: keep-alive
Host: 127.0.0.1:8080
Referer: http://127.0.0.1:8080/
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36
sec-ch-ua: "Not_A Brand";v="8", "Chromium";v="120", "Google
Chrome";v="120"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
```

2. Server response header

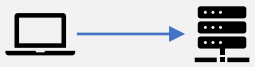
```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.11.5
Date: Wed, 17 Jan 2024 21:45:21 GMT
Content-type: text/html; charset=utf-8
Content-Length: 301
```

3. Server response body

```
<!DOCTYPE HTML>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Directory listing for /.conda/</title>
</head>
<body>
<h1>Directory listing for /.conda/</h1>
<hr>
<ul>
<li><a href="aau_token">aau_token</a></li>
<li><a href="environments.txt">environments.txt</a></li>
</ul>
<hr>
</body>
</html>
```

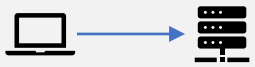
BTW: Common Crawl provides billions of web pages *and* the request/response headers

A closer look at the HTTP request

Browser to the server 

```
GET /.conda/ HTTP/1.1
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,a
pplication/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cache-Control: max-age=0
Connection: keep-alive
Host: 127.0.0.1:8080
Referer: http://127.0.0.1:8080/
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/120.0.0.0 Safari/537.36
sec-ch-ua: "Not_A Brand";v="8", "Chromium";v="120", "Google Chrome";v="120"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
```

A closer look at the HTTP response

Server to the browser 

Header

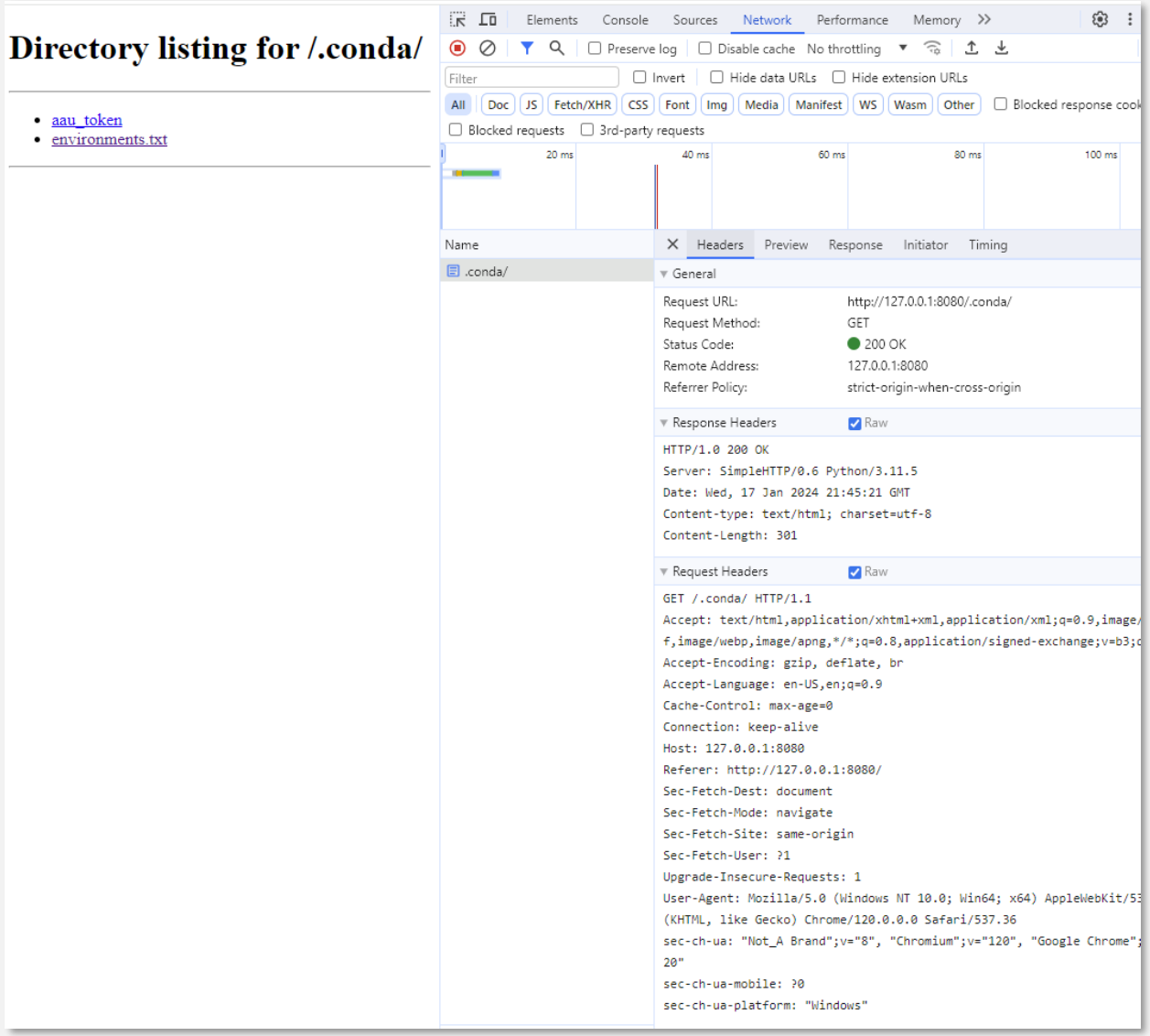
```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.11.5
Date: Wed, 17 Jan 2024 21:45:21 GMT
Content-type: text/html; charset=utf-8
Content-Length: 301
```

Body

```
<!DOCTYPE HTML>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Directory listing for /.conda/</title>
</head>
<body>
<h1>Directory listing for /.conda/</h1>
<hr>
<ul>
<li><a href="aau_token">aau_token</a></li>
<li><a
href="environments.txt">environments.txt</a></li>
</ul>
<hr>
</body>
</html>
```

What is exchanged between your browser and the server?

The HTTP protocol is a text based protocol. It is easy to parse, easy to debug and easy to wrap in libraries of any language



Closer look at HTTP methods

```
::ffff:127.0.0.1 - - [17/Jan/2024 15:25:43] "GET / HTTP/1.1" 200 -
::ffff:127.0.0.1 - - [17/Jan/2024 15:27:24] "GET /.conda/ HTTP/1.1" 200 -
::ffff:127.0.0.1 - - [17/Jan/2024 15:27:29] "GET /.conda/environments.txt HTTP/1.1" 200 -
```

GET

The HTTP protocol defines some methods, such as GET, POST, PUT, DELETE. GET translates to “read a resources at the given route”

The http protocol provides four methods

Method	CRUD	Web developer description	Data scientist description
POST	**C**reate	Create new resource	Can submit data, such as inputs to a model
GET	**R**ead	Retrieves web pages at a given route	Normal web request
PUT	**U**pdate	Update a resource	Not relevant
DELETE	**D**elete	Remove a resource	Not relevant

HTTP methods: GET and POST

We have seen that `GET` is the same as a standard web request

`http://somewebsite.com/section_a/item_b?param1=value&param2=value`

If we are building a prediction service, the inputs to the model can just be the arguments `param1` and `param2`?

A more generally accepted method is to use `POST`. **`POST` allows clients to submit a JSON* object as a "payload,"** along with the url.

```
http://somewebsite.com/section_a/item_b
{param1:value, param2:value}
```

Parameters encoded in the URL, as with `GET`, have space limits and often need to encode special characters. `POST` removes such limitations.

*`POST` is not limited to JSON, it can be any data

HTTP libraries

Practically every language has libraries for writing http clients and servers

You can even use the command line to access a simple http resource:

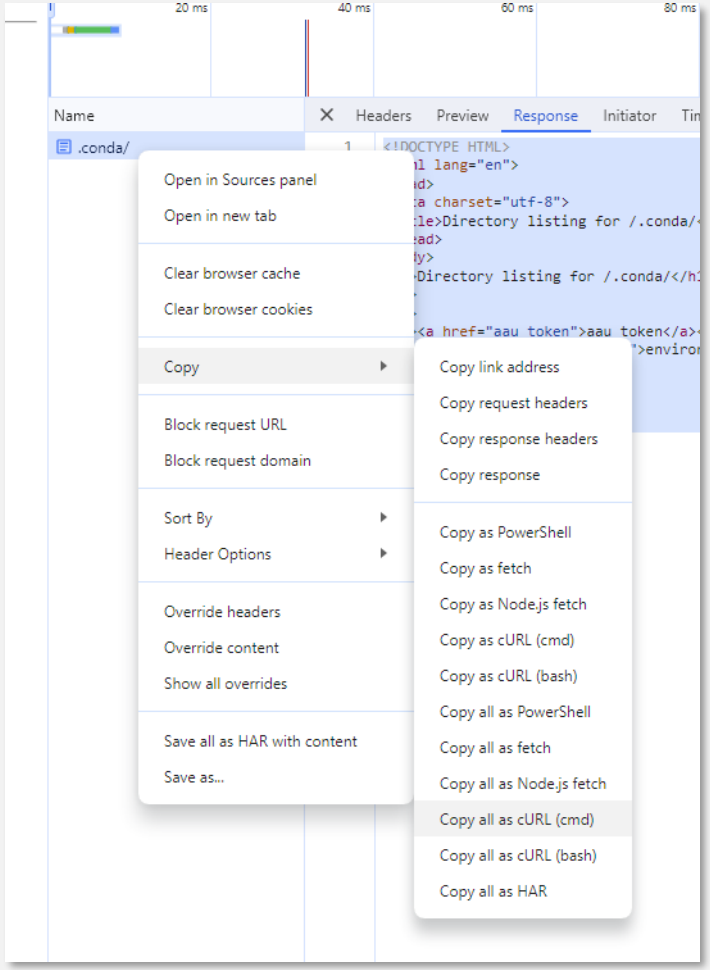
`curl http://127.0.0.1:8080/.conda/`

```
C:\Users\shahb>curl localhost:8080/.conda/
<!DOCTYPE HTML>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Directory listing for /.conda/</title>
</head>
<body>
<h1>Directory listing for /.conda/</h1>
<hr>
<ul>
<li><a href="aau_token">aau_token</a></li>
<li><a href="environments.txt">environments.txt</a></li>
</ul>
<hr>
</body>
</html>
```

Practically every language has libraries for writing http clients and servers

You can even use the command line to access a not so simple http resource:

```
curl "http://127.0.0.1:8080/.conda/" ^
  -H "Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/ap
ng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7" ^
  -H "Accept-Language: en-US,en;q=0.9" ^
  -H "Cache-Control: max-age=0" ^
  -H "Connection: keep-alive" ^
  -H "Referer: http://127.0.0.1:8080/" ^
  -H "Sec-Fetch-Dest: document" ^
  -H "Sec-Fetch-Mode: navigate" ^
  -H "Sec-Fetch-Site: same-origin" ^
  -H "Sec-Fetch-User: ?1" ^
  -H "Upgrade-Insecure-Requests: 1" ^
  -H "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36" ^
  -H "sec-ch-ua: ^\^"Not_A Brand^\^";v=^\^"8^\^", ^\^"Chromium^\^";v=^\^"120^\^",
^\^"Google Chrome^\^";v=^\^"120^\^"" ^
  -H "sec-ch-ua-mobile: ?0" ^
  -H "sec-ch-ua-platform: ^\^"Windows^\^"" ^
  --compressed
```



cURL is an extremely versatile tool

Normal `GET` request

```
curl http://somewebsite.com/some_resource
```

`POST` request with a JSON payload

```
curl -X POST \
  -H “Content-Type: application/json” \
  -d “{param1:val, param2:val}” http://somewebsite.com/some_resource
```

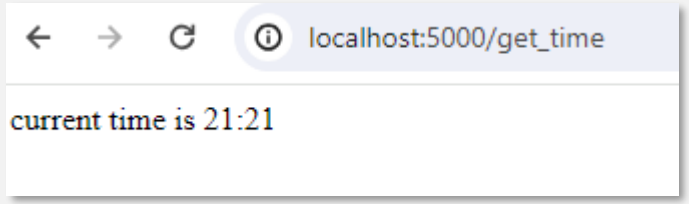
Web services

If HTTP is supported by every language, every operating system, many browsers, even the command line, isn't it also a **great tool to exchange data**?

That's what web services are!

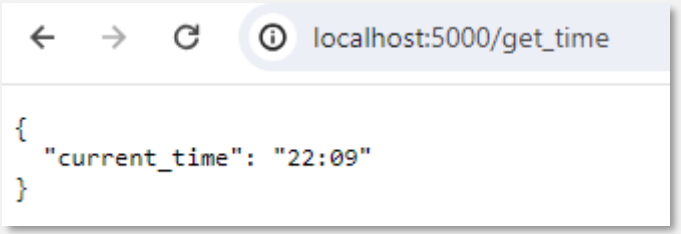
Web services: serve data, instead of web pages (… but text?)

```python
#serve text
from datetime import datetime
from flask import Flask, request, send_from_directory

app = Flask(__name__)

@app.get('/get_time')
def get_time():
    t = f'"current_time":"{datetime.now().strftime("%H:%M")}"'
    return t

if __name__=='__main__':
    app.run(debug=True, port=5000) #host='0.0.0.0' <= Listen to all traffic, not just local
```



Web services: serve JSON

```python
#serve json
from datetime import datetime
from flask import Flask, request, send_from_directory, jsonify

app = Flask(__name__)

@app.get('/get_time')
def get_time():
    d = {'current_time':datetime.now().strftime("%H:%M")}
    return jsonify(d)

if __name__=='__main__':
    app.run(debug=True, port=5000)
```



A web services *system* produces and consumes JSON over HTTP

```python
from flask import Flask, request, send_from_directory, jsonify

app = Flask(__name__)

@app.post('/get_churn_probability')
def get_churn_probability():

    client_properties = request.get_json()
    #if 'UC_GRAD' not in client_properties:
    #    pass throw error

    # Our churn model is fake
    if client_properties['uc_grad']:
        return jsonify({'churn_prob':0.34})
    else:
        return jsonify({'churn_prob':0.87})

if __name__=='__main__':
    app.run(debug=True, port=5000)
```

Request the 'get_churn_probability' service (recall that it is a POST service)

```
import requests
```

```
HOST = 'localhost'
PORT = 5000
```

```
good_client  = {'gender':'male', 'age':23, 'uc_grad':False}
great_client = {'gender':'male', 'age':23, 'uc_grad':True}
```

```
%%time

response = requests.post(url=f'http://{HOST}:{PORT}/get_churn_probability', json=good_client)
```

```
CPU times: total: 0 ns
Wall time: 2.03 s
```

```
response
```

```
<Response [200]>
```

```
response.json()
```

```
{'churn_prob': 0.87}
```

```
requests.post(url=f'http://{HOST}:{PORT}/get_churn_probability', json=great_client).json()
```

```
{'churn_prob': 0.34}
```

See notebook 075_web_services_under_the_hood/consume_services.ipynb

In-class exercise

Please write a small service which uses a POST method to accept two numbers, add them and return the result.

You should accept a JSON object like this: `{num1:2, num2:3}`

Be prepared to share your IP (ifconfig/ipconfig) and port with me so I (and others) can test your service.

Appendix

Syntax highlighting