NYU | Center for Data Science

# DS-UA 112
# Introduction to Data Science

## Lecture 13
## SQL I – Working with Databases

# Announcements

- Homework 3
  - Due Friday October 18
- Project 1
  - Extended to Sunday October 27
- Midterm
  - Wednesday October 23 4:55-6:10
  - Pencil and Paper with Cheat-Sheets
  - Section and Office Hours
  - Practice Exam

# Review (DEMO)

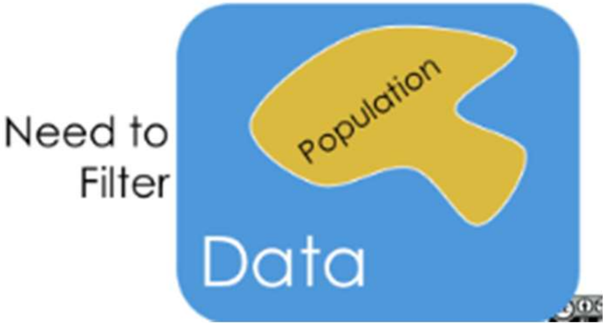
- **Granularity**
  - How fine/coarse is each datum?
- **Scope**
  - How (in)complete are the data?
- **Temporality**
  - How are the data situated in time?
- **Faithfulness**
  - How accurately do the data describe the world?

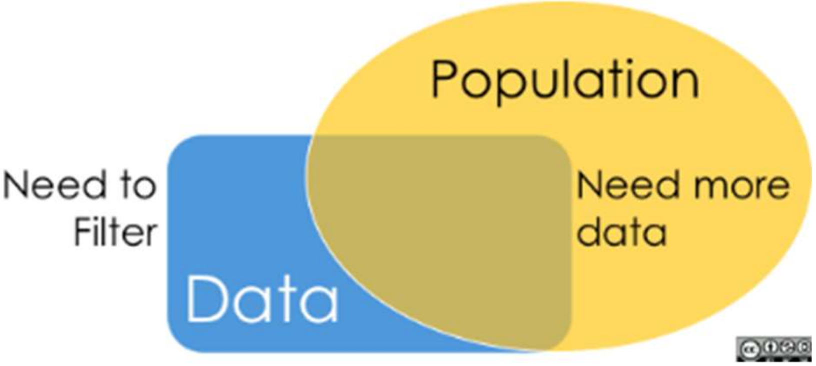
# Review

- The ***granularity*** of your data is what each record represents… is it coarse or fine?
  - What does a record represent?
  - Do all records capture granularity at the same level? If the data were aggregated, how was the aggregation performed?
    - Sampling
    - Averging
  - What kinds of aggregations can we perform on the data? In general, how do we change the granularity?

# Review

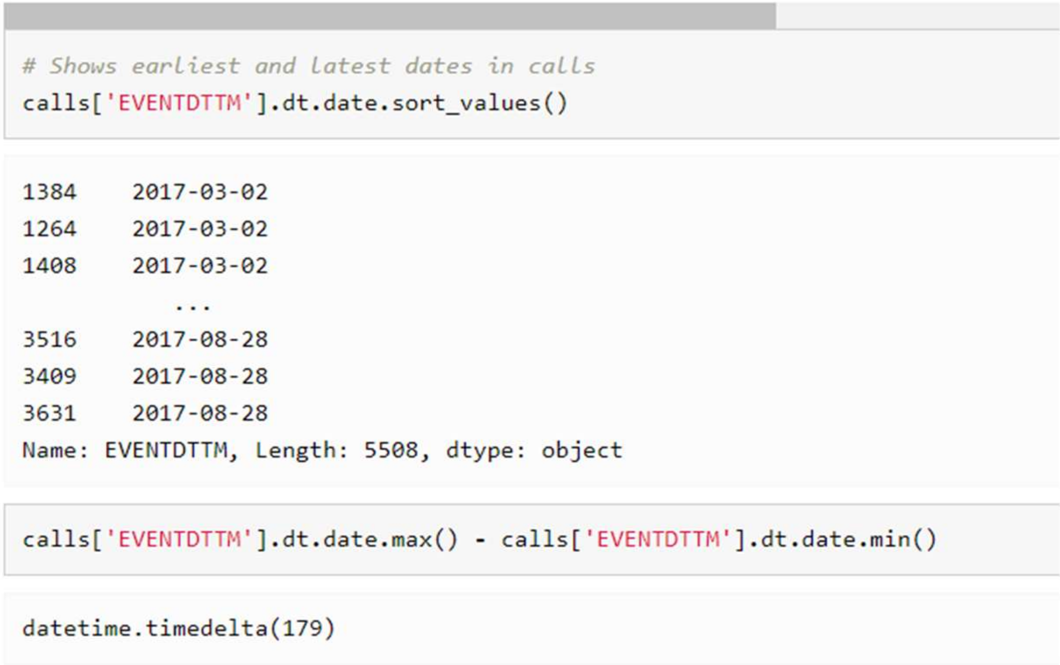
- The **scope** of the dataset refers to the coverage of the dataset in relation to what we are interested in analyzing.
  - Geographic Scope?





# Review

- The ***temporality*** refers to the date and time fields in the dataset.
  - What is the meaning of the date and time fields in the dataset?
  - What representation do the date and time fields have in the data?
  - Are there strange timestamps that might represent null values?



```
# Shows earliest and latest dates in calls
calls['EVENTDTTM'].dt.date.sort_values()
```

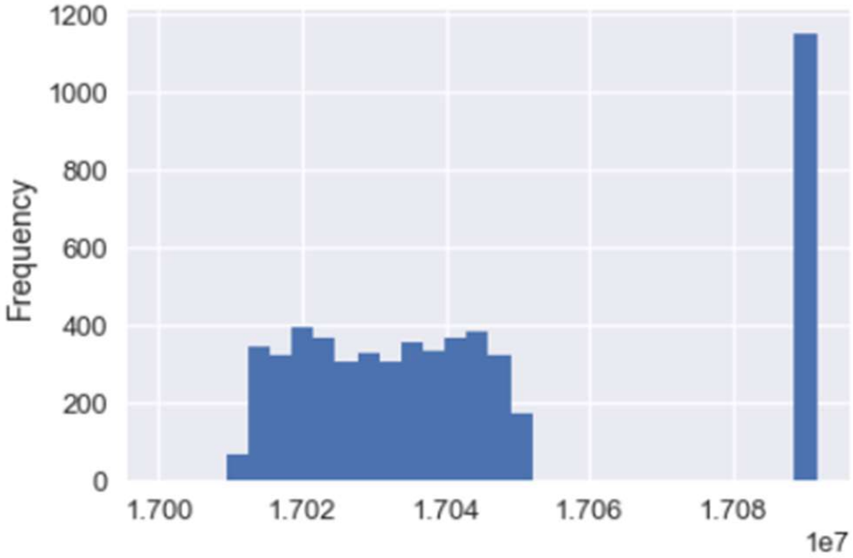
```
1384    2017-03-02
1264    2017-03-02
1408    2017-03-02
           ...
3516    2017-08-28
3409    2017-08-28
3631    2017-08-28
Name: EVENTDTTM, Length: 5508, dtype: object
```

```
calls['EVENTDTTM'].dt.date.max() - calls['EVENTDTTM'].dt.date.min()
```

```
datetime.timedelta(179)
```

# Review

- We describe a dataset as ***faithful*** if we believe it accurately captures reality.
  - Unrealistic or incorrect values
  - Violations of obvious dependencies
  - Hand-entered data
  - Clear signs of data falsification



# Agenda

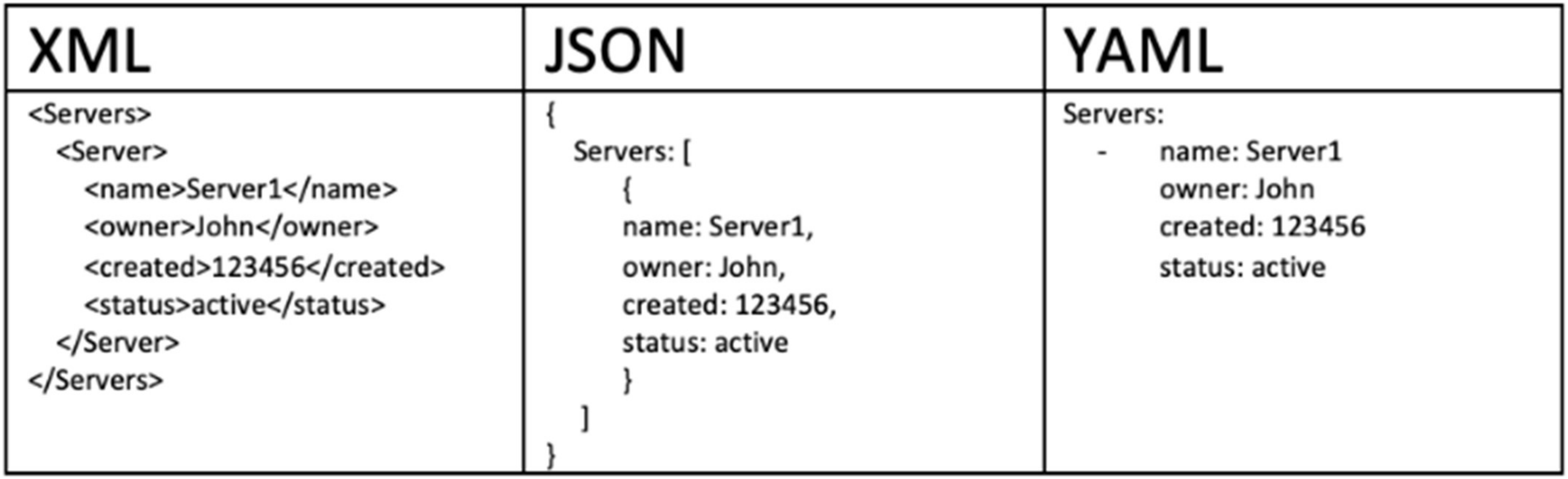
- Lessons
  - Connecting to Websites
  - SQL for Databases
- Demos
  - Police Reports
  - Wikipedia
- Questions

## Objectives

- Application Programming Interfaces
  - What file formats do we need for Websites?
  - Explain a request-response protocol
- Structure Query Language
  - Understanding commands for table manipulations
- Readings:
  - Nolan 7.1, 9
  - Grus Appendix

# Data Formats for Websites

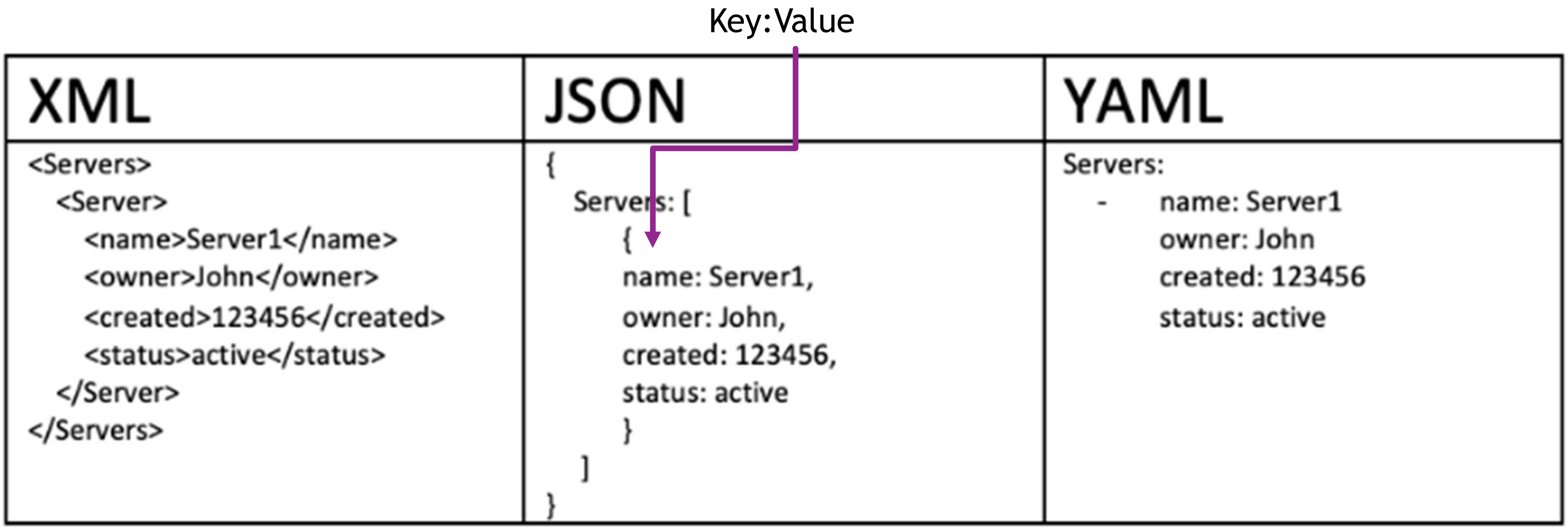
- Descriptive
- Extensible
- Human and Machine Readable



| XML | JSON | YAML |
|---|---|---|
| &lt;Servers&gt;<br>&lt;Server&gt;<br>&lt;name&gt;Server1&lt;/name&gt;<br>&lt;owner&gt;John&lt;/owner&gt;<br>&lt;created&gt;123456&lt;/created&gt;<br>&lt;status&gt;active&lt;/status&gt;<br>&lt;/Server&gt;<br>&lt;/Servers&gt; | {<br>Servers: [<br>{<br>name: Server1,<br>owner: John,<br>created: 123456,<br>status: active<br>}<br>]<br>} | Servers:<br>- name: Server1<br>owner: John<br>created: 123456<br>status: active |

# JavaScript Object Notation

- Key: Value
- Value is Array of
  - string, number, Boolean, null

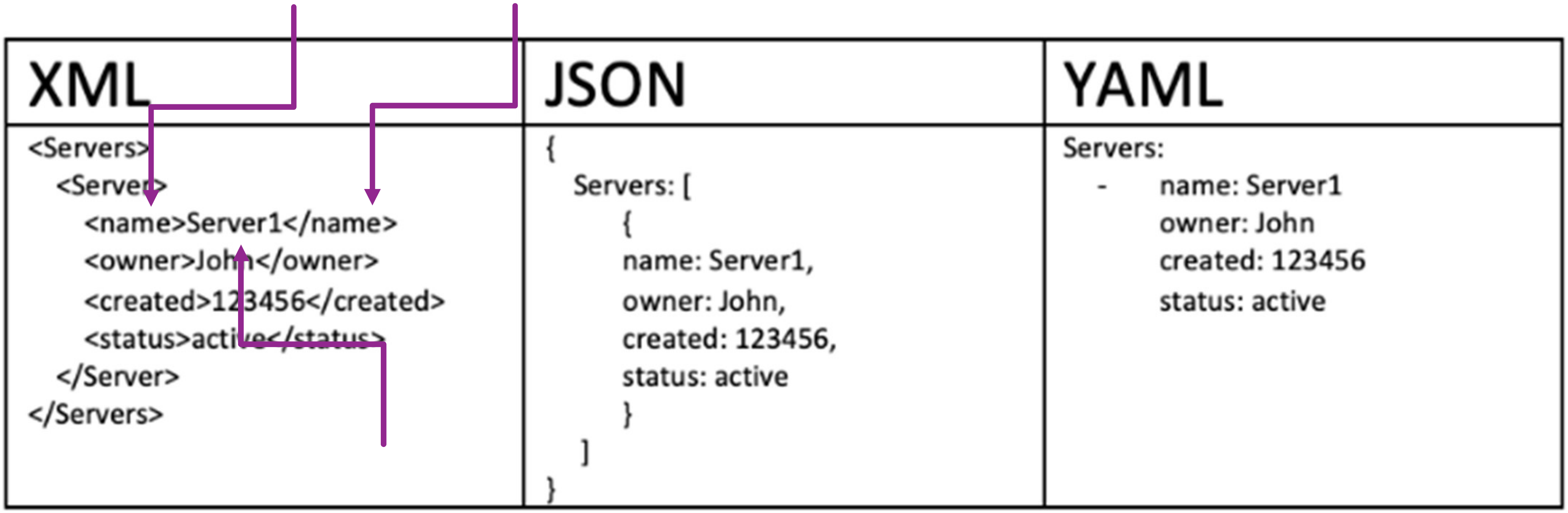


Key:Value

| XML | JSON | YAML |
|---|---|---|
| <Servers><br>  <Server><br>    <name>Server1</name><br>    <owner>John</owner><br>    <created>123456</created><br>    <status>active</status><br>  </Server><br></Servers> | {<br>  Servers: [<br>    {<br>    name: Server1,<br>    owner: John,<br>    created: 123456,<br>    status: active<br>    }<br>  ]<br>} | Servers:<br>  -    name: Server1<br>       owner: John<br>       created: 123456<br>       status: active |

# eXtensible Markup Language

- Start Tag
- End Tag
- Content along with other nodes

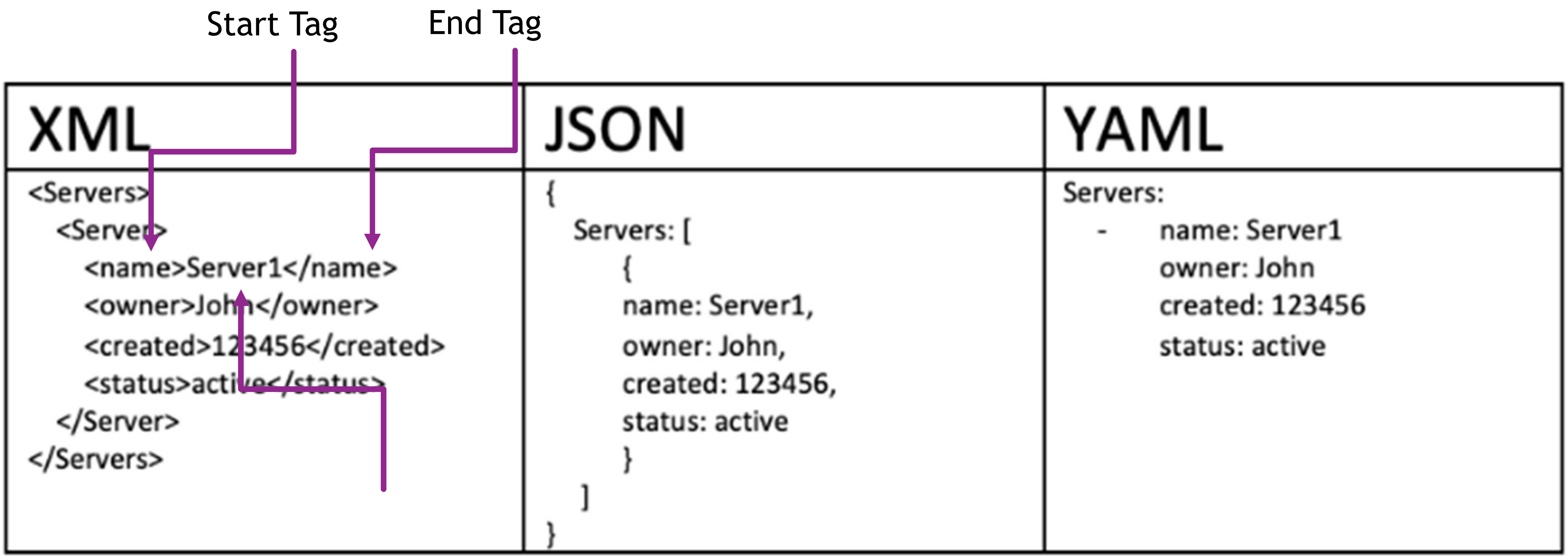


Start Tag
End Tag
Content

| XML | JSON | YAML |
|---|---|---|
| <Servers><br>  <Server><br>    <name>Server1</name><br>    <owner>John</owner><br>    <created>123456</created><br>    <status>active</status><br>  </Server><br></Servers> | {<br>  Servers: [<br>    {<br>    name: Server1,<br>    owner: John,<br>    created: 123456,<br>    status: active<br>    }<br>  ]<br>} | Servers:<br>  -    name: Server1<br>        owner: John<br>        created: 123456<br>        status: active |

# eXtensible Markup Language

- Properly nested instead each other
- If content empty, then <tagname/> enough



Start Tag

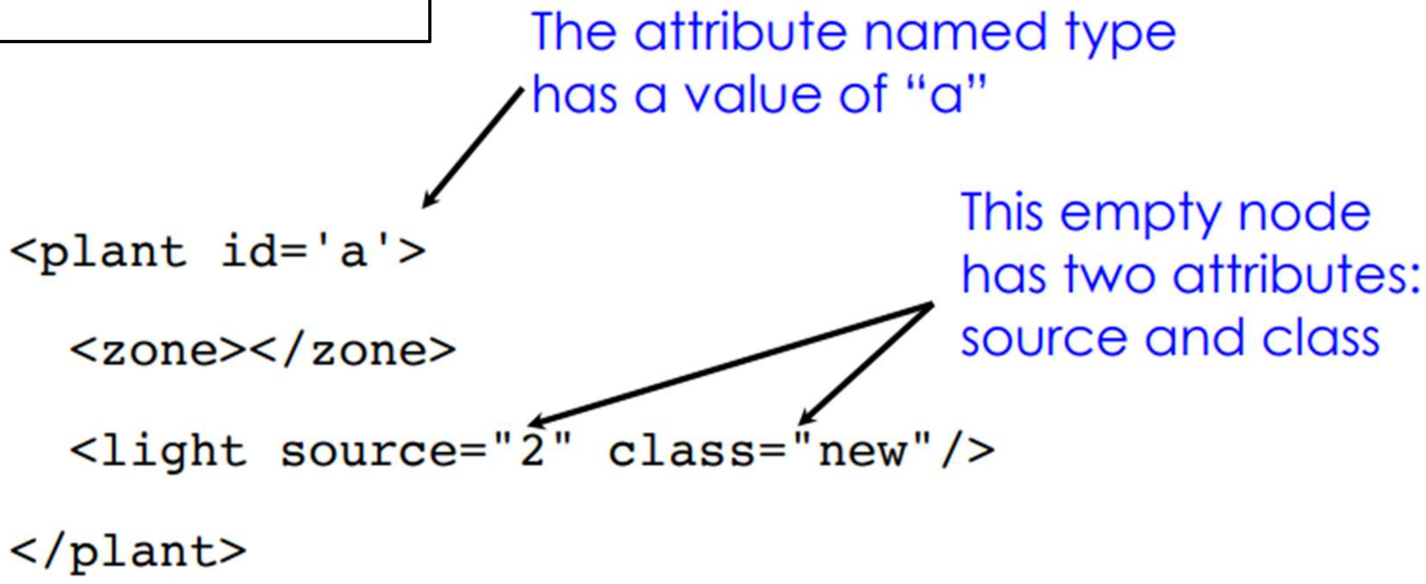
End Tag

| XML | JSON | YAML |
|---|---|---|
| <Servers><br><Server><br><name>Server1</name><br><owner>John</owner><br><created>123456</created><br><status>active</status><br></Server><br></Servers> | {<br>Servers: [<br>{<br>name: Server1,<br>owner: John,<br>created: 123456,<br>status: active<br>}<br>]<br>} | Servers:<br>- name: Server1<br>owner: John<br>created: 123456<br>status: active |

Content

# eXtensible Markup Language

- attributes must appear in quotes such as name = "value“
- <!-- this is a comment -->

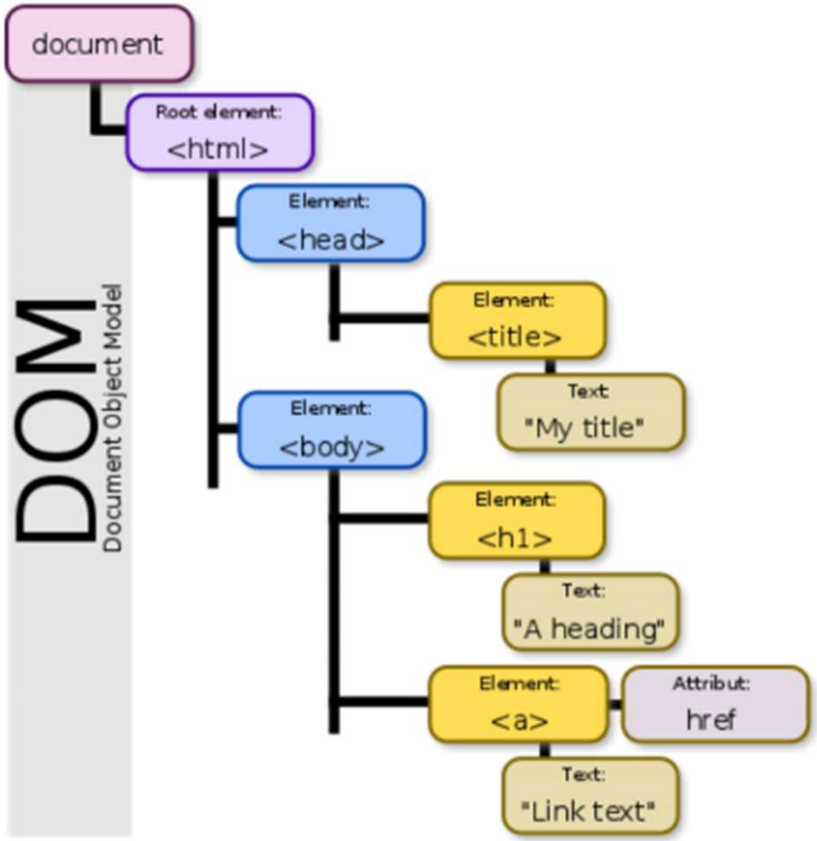


The attribute named type has a value of “a”

```
<plant id='a'>
  <zone></zone>
  <light source="2" class="new"/>
</plant>
```

This empty node has two attributes: source and class

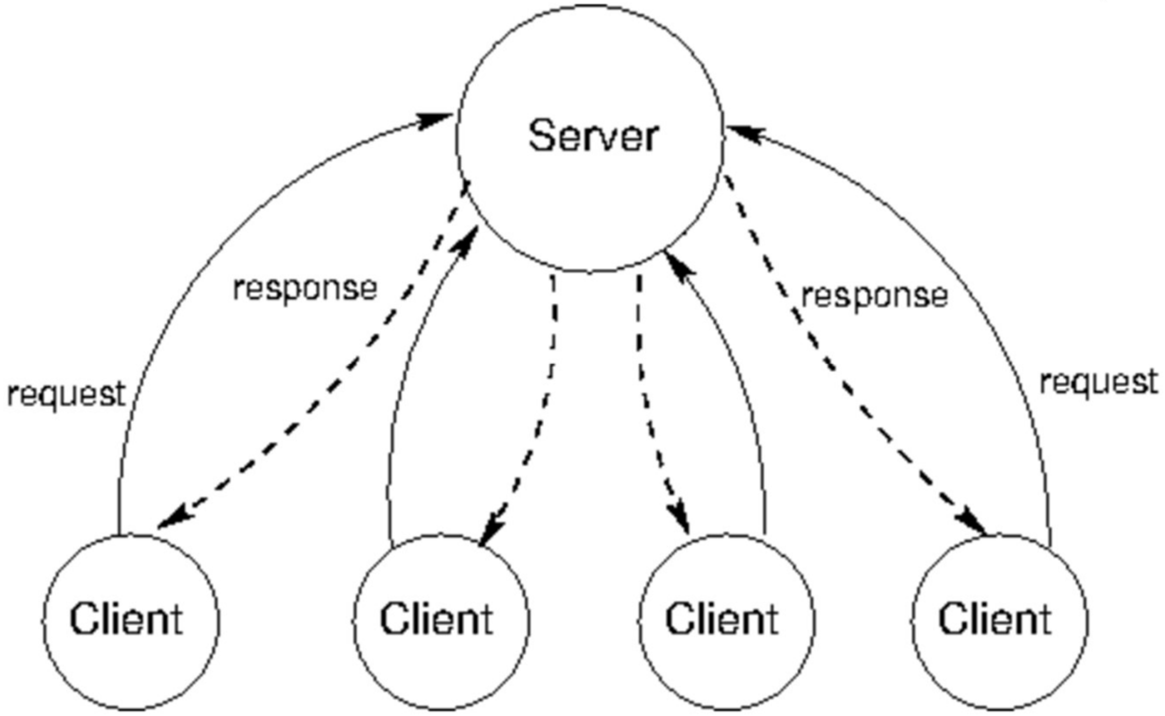
# DOM: Document Object Model (DEMO)



- There is only one root in the tree, and all other nodes are contained within it.
- We refer to relationships between nodes: parents, children, siblings, ancestors, descendants
- The terminal nodes in a tree are also known as leaf nodes. Content always falls in a leaf node.

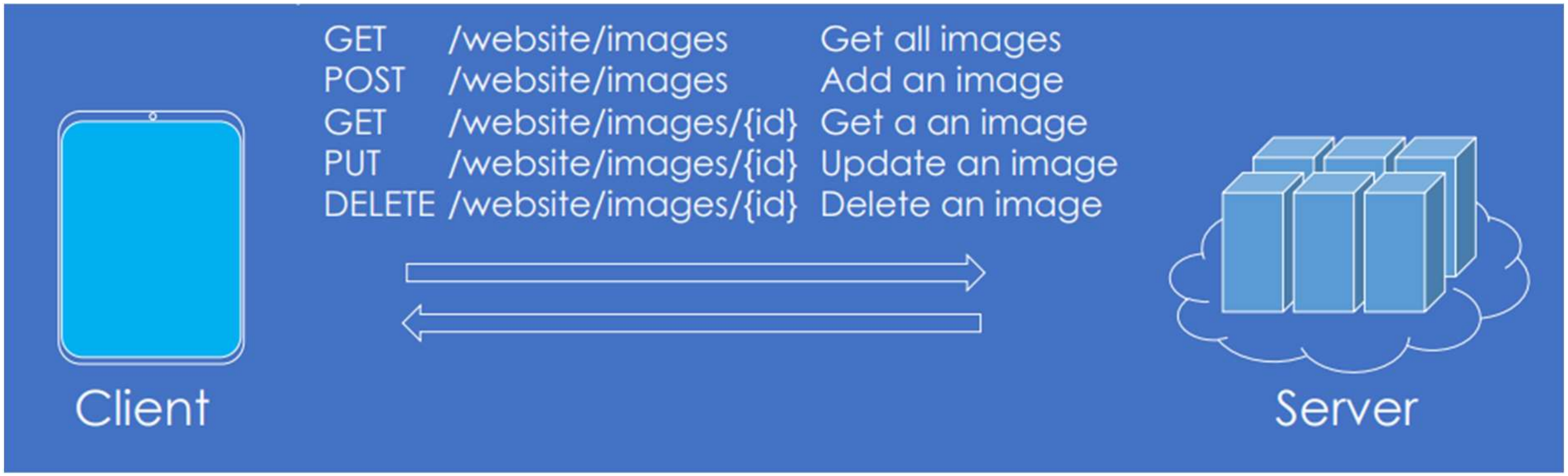
# REST – Representational State Transfer

- Widely accessible, efficient, and extensible web services
  - Client-Server with Response-Request



# REST – Representational State Transfer

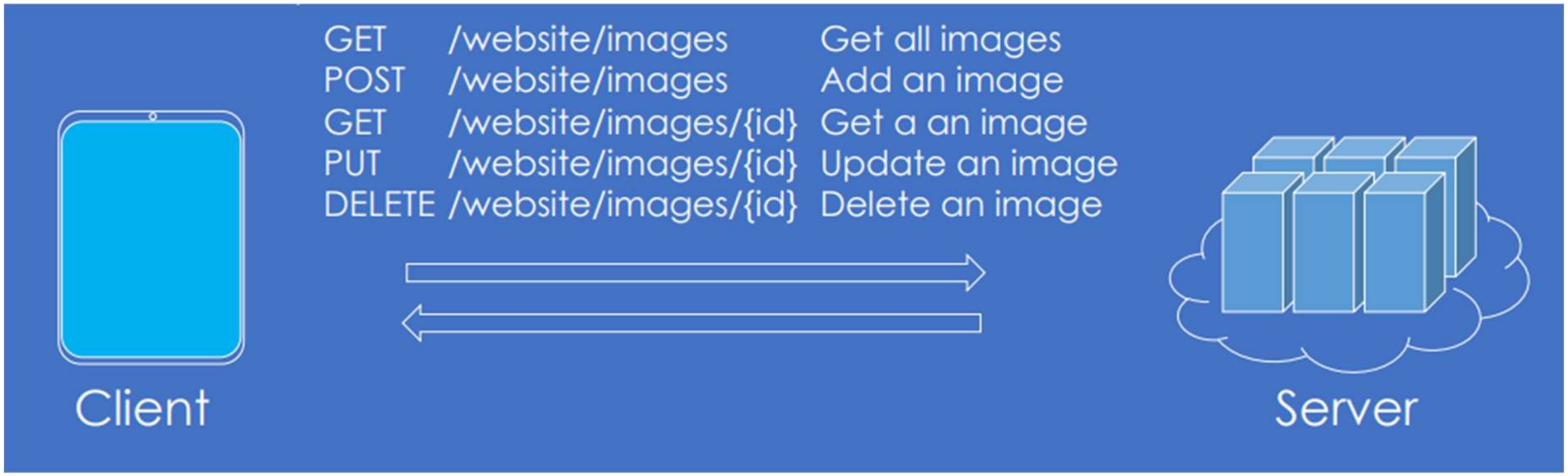
- Widely accessible, efficient, and extensible web services
  - Client-Server with Response-Request
  - HTTP provides approach to REST API



| | | |
|---|---|---|
| GET | /website/images | Get all images |
| POST | /website/images | Add an image |
| GET | /website/images/{id} | Get a an image |
| PUT | /website/images/{id} | Update an image |
| DELETE | /website/images/{id} | Delete an image |

# REST – Representational State Transfer

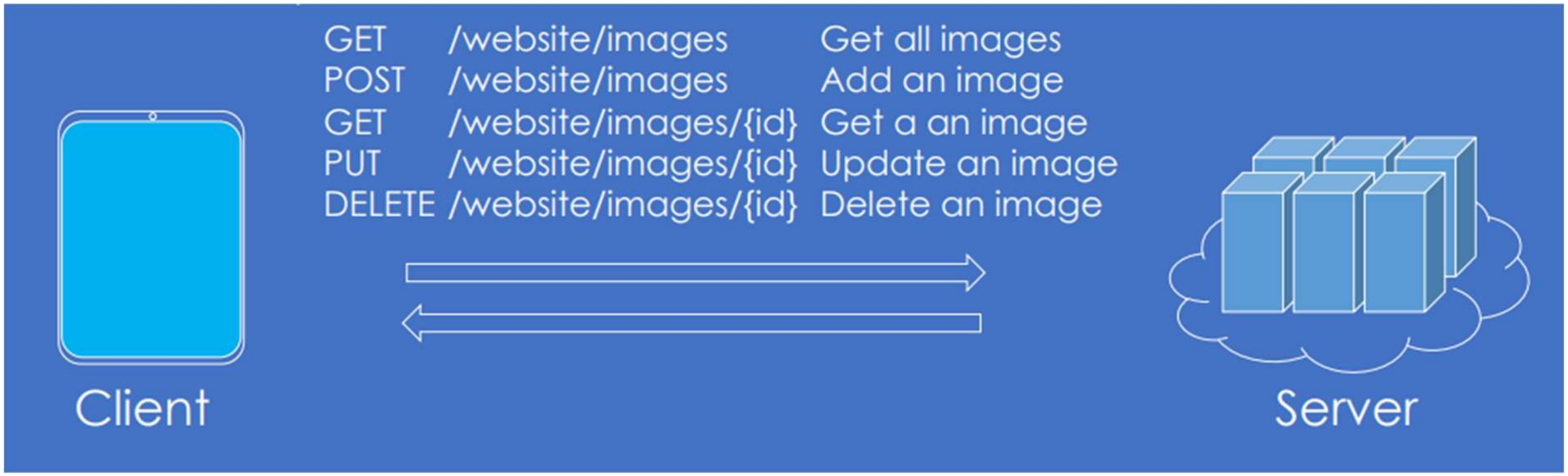
- Guidelines for API
  - Uniform Interface
  - Separation Client-Server with layers in between



| | | |
|---|---|---|
| GET | /website/images | Get all images |
| POST | /website/images | Add an image |
| GET | /website/images/{id} | Get a an image |
| PUT | /website/images/{id} | Update an image |
| DELETE | /website/images/{id} | Delete an image |

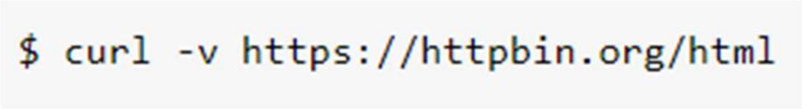
# REST – Representational State Transfer

- Guidelines for API
  - Uniform Interface
  - Separation Client-Server with layers in between
  - Stateless
  - Cacheable



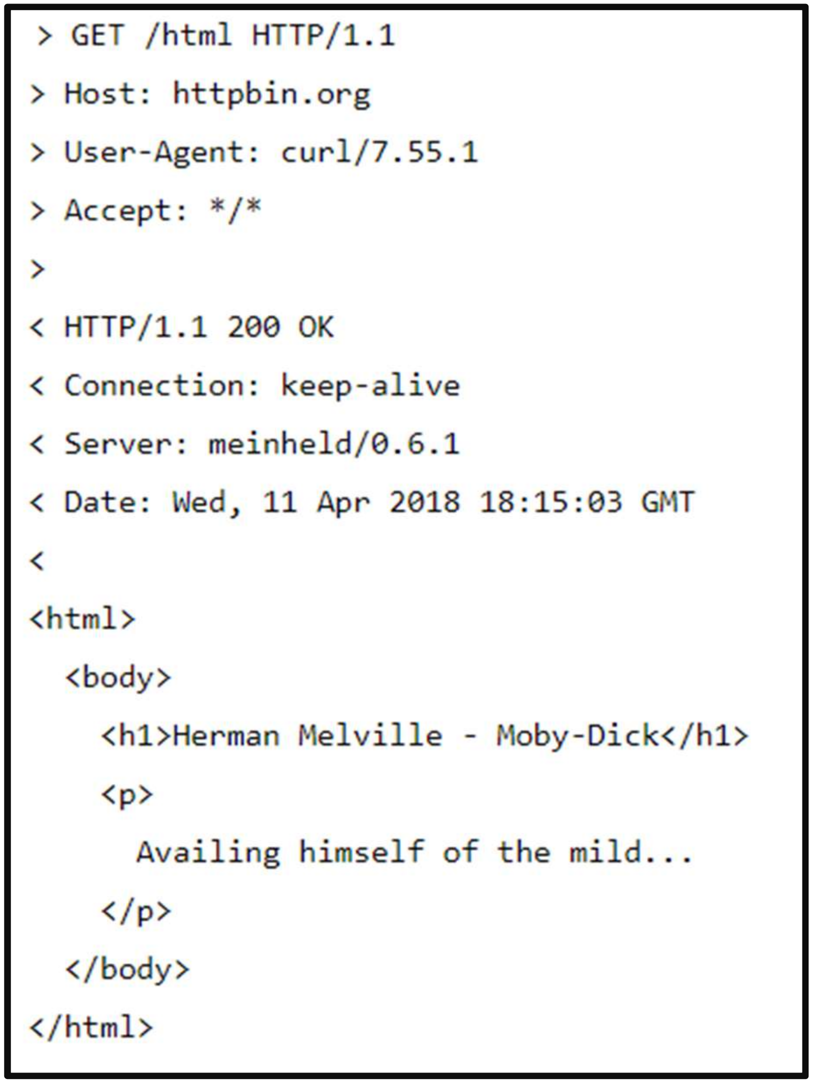
| | | |
|---|---|---|
| GET | /website/images | Get all images |
| POST | /website/images | Add an image |
| GET | /website/images/{id} | Get a an image |
| PUT | /website/images/{id} | Update an image |
| DELETE | /website/images/{id} | Delete an image |

# Command Line



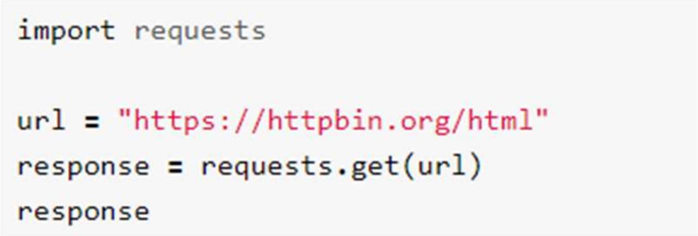
```
$ curl -v https://httpbin.org/html
```

- Hyper Text Transfer Protocol
  - GET
  - POST
  - PUT
  - DELETE



```
> GET /html HTTP/1.1
> Host: httpbin.org
> User-Agent: curl/7.55.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Connection: keep-alive
< Server: meinheld/0.6.1
< Date: Wed, 11 Apr 2018 18:15:03 GMT
<
<html>
  <body>
    <h1>Herman Melville - Moby-Dick</h1>
    <p>
      Availing himself of the mild...
    </p>
  </body>
</html>
```

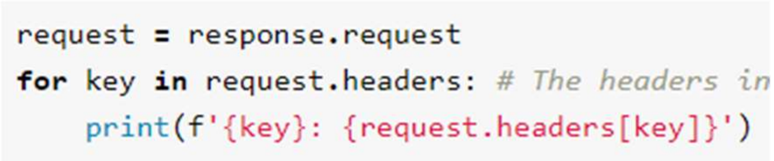
# requests Package



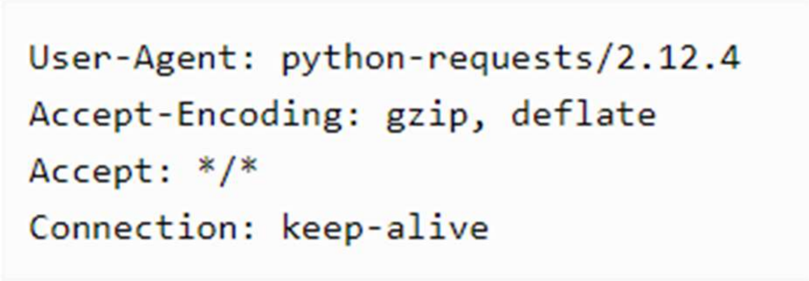
```python
import requests

url = "https://httpbin.org/html"
response = requests.get(url)
response
```

- Hyper Text Transfer Protocol
  - GET
  - POST
  - PUT
  - DELETE

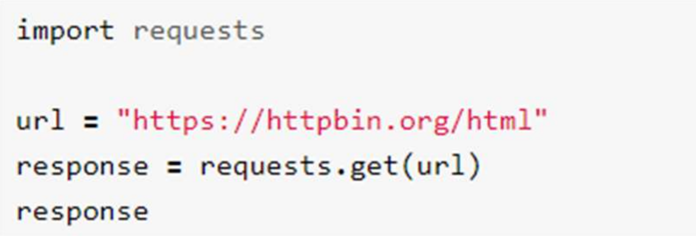


```python
request = response.request
for key in request.headers: # The headers in
    print(f'{key}: {request.headers[key]}')
```



```
User-Agent: python-requests/2.12.4
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
```

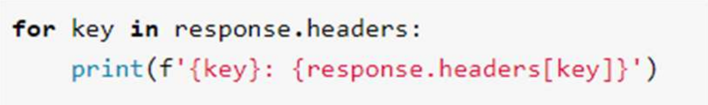
# requests Package



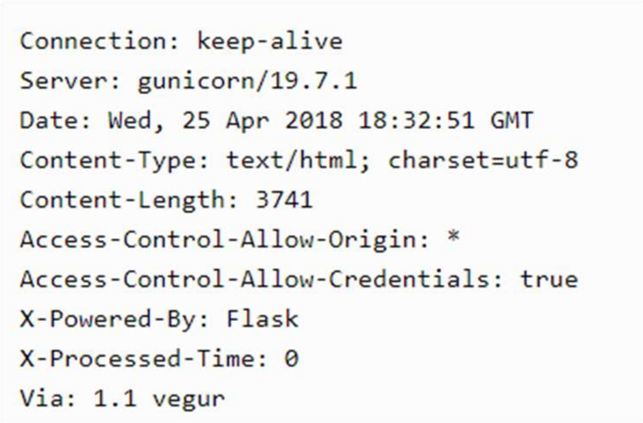
```python
import requests

url = "https://httpbin.org/html"
response = requests.get(url)
response
```

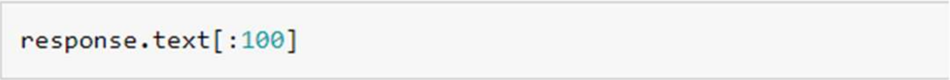
- Hyper Text Transfer Protocol
  - GET
  - POST
  - PUT
  - DELETE



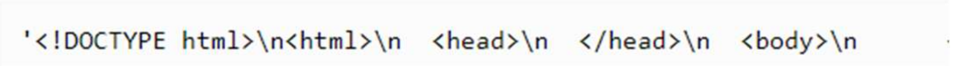
```python
for key in response.headers:
    print(f'{key}: {response.headers[key]}')
```



```
Connection: keep-alive
Server: gunicorn/19.7.1
Date: Wed, 25 Apr 2018 18:32:51 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 3741
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
X-Powered-By: Flask
X-Processed-Time: 0
Via: 1.1 vegur
```



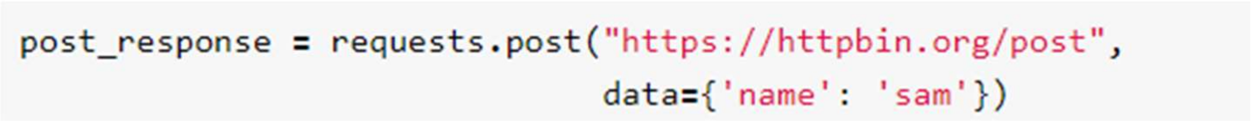
```python
response.text[:100]
```



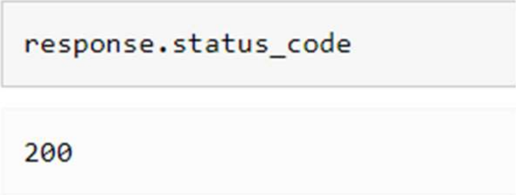
```
'<!DOCTYPE html>\n<html>\n  <head>\n  </head>\n  <body>\n
```

# requests Package

- Hyper Text Transfer Protocol
  - GET
  - POST
  - PUT
  - DELETE

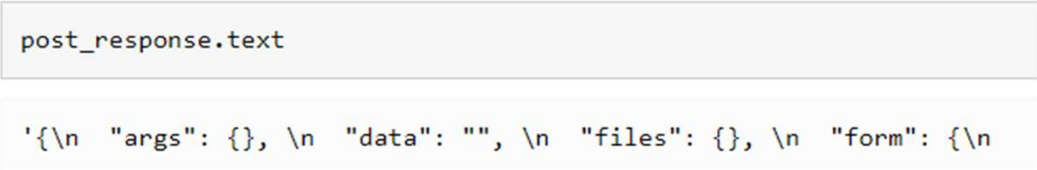


```python
post_response = requests.post("https://httpbin.org/post",
                              data={'name': 'sam'})
```



```python
response.status_code
```

```
200
```



```python
post_response.text
```

```
'{\n  "args": {}, \n  "data": "", \n  "files": {}, \n  "form": {\n
```

# Status Codes

- Hyper Text Transfer Protocol
  - GET
  - POST
  - PUT
  - DELETE

- **100s** - Informational: More input is expected from client or server *(e.g. 100 Continue, 102 Processing)*
- **200s** - Success: The client's request was successful *(e.g. 200 OK, 202 Accepted)*
- **300s** - Redirection: Requested URL is located elsewhere; May need user's further action *(e.g. 300 Multiple Choices, 301 Moved Permanently)*
- **400s** - Client Error: Client-side error *(e.g. 400 Bad Request, 403 Forbidden, 404 Not Found)*
- **500s** - Server Error: Server-side error or server is incapable of performing the request *(e.g. 500 Internal Server Error, 503 Service Unavailable)*

# Web Scraping (DEMO)

- Don’t violate terms of use for the service or data
- Scraping can cause result in degraded services for others
  - Many services are optimized for human user access patterns
  - Requests can be parallelized/distributed to saturate server
  - Each query may result in many database requests
- How to scrape ethically:
  - Used documented REST APIs – read terms of service
  - Examine at robots.txt
  - Throttle request rates (sleep)
  - Avoid getting NYU blocked from websites & services

# Take-Aways

- File Formats for Websites
  - JSON, YAML
  - XML, HTML
- DOM
- REST API's
  - GET, POST, PUT, DELETE