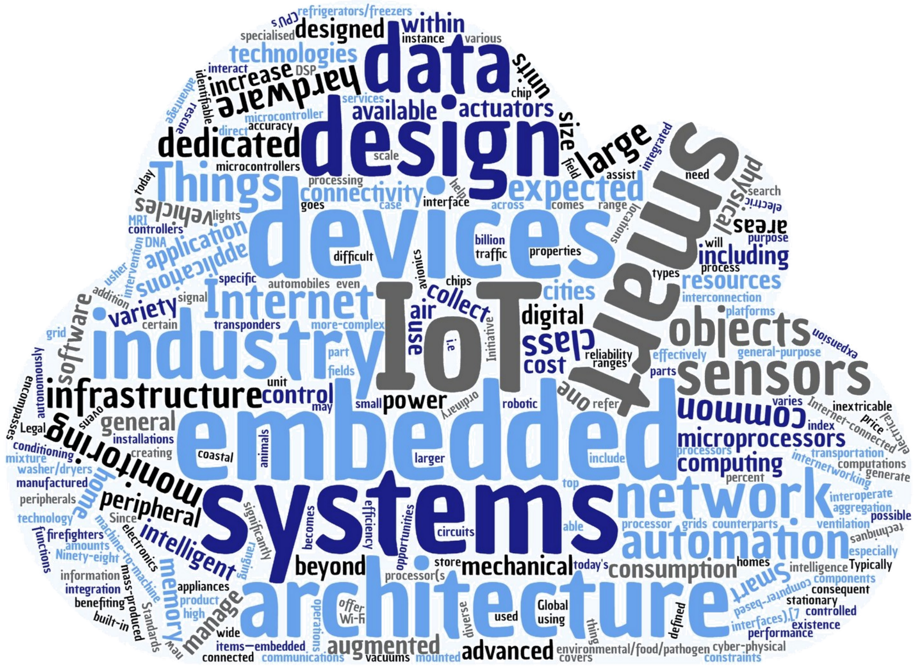
Associate Professorship of Embedded Systems and Internet of Things
Department of Electrical and Computer Engineering
Technical University of Munich

Lecture
IoT Remote Lab

02 - Architectures and Payloads

Ege Korkan

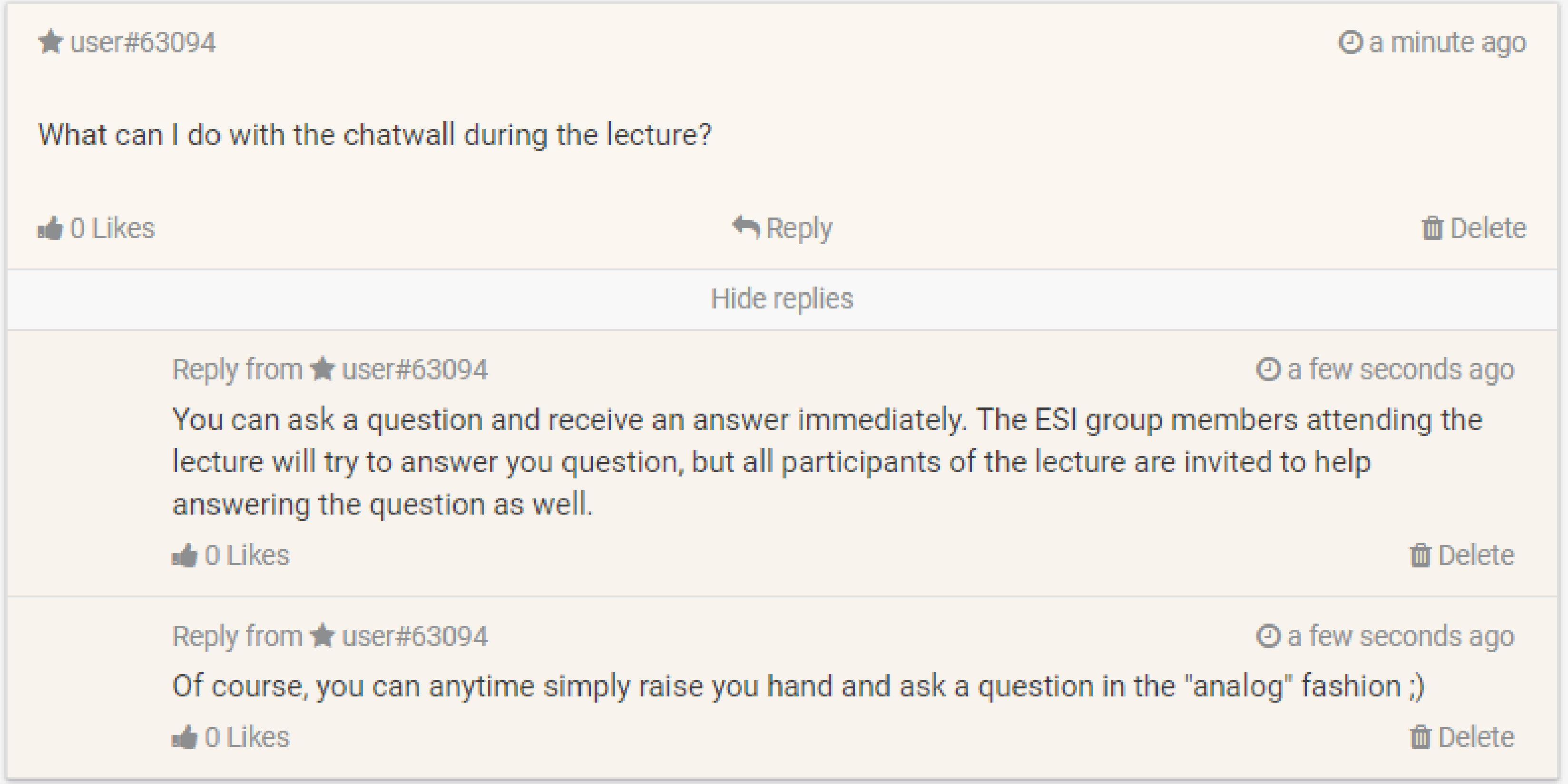


Zoom Guidelines

- The lectures and tutor sessions happen on Zoom meetings following the link sent to you via email.
- The lecture will be recorded and uploaded in a way that is accessible only to course participants
- The recording will be paused when a student speaks.
- Tutor sessions will not be recorded.
- Participatition to Zoom sessions is optional
- You can choose a random string for your name
- The Zoom chat will not be recorded and we will not save the chat.
- All the participants except the lecturer is muted. The participants are **not** free to unmute. You must go to participants, click the hand icon to raise your hand.
- You can also do other things, like asking me to go slower. I have a separate window where I look at the requests from the participants.
- You can ask quick questions in Zoom chat or use the Tweedback link provided in each session.

Tweedback for Real-time Q&A During the Lecture

With Tweedback, you can ask questions during the lecture in real-time and anonymously, if you want.
To access the Tweedback chatwall, you access the address tum.tweedback.de/****, where **** stands for the session ID of the specific day, such as https://tweedback.de/kqr7 for today



★ user#63094 — a minute ago

What can I do with the chatwall during the lecture?

Reply from ★ user#63094 — a few seconds ago

You can ask a question and receive an answer immediately. The ESI group members attending the lecture will try to answer you question, but all participants of the lecture are invited to help answering the question as well.

Reply from ★ user#63094 — a few seconds ago

Of course, you can anytime simply raise you hand and ask a question in the "analog" fashion ;)

Participants

- The Zoom invitation is valid for every week
- If you are not planning to pursue the course, please deregister now!
- Some checks:
  - Can you see the Moodle contents?
  - Can you see the videos on Moodle and Panopto?
  - Have you set up your development environment? If not, there is a tutor session at the end of this session

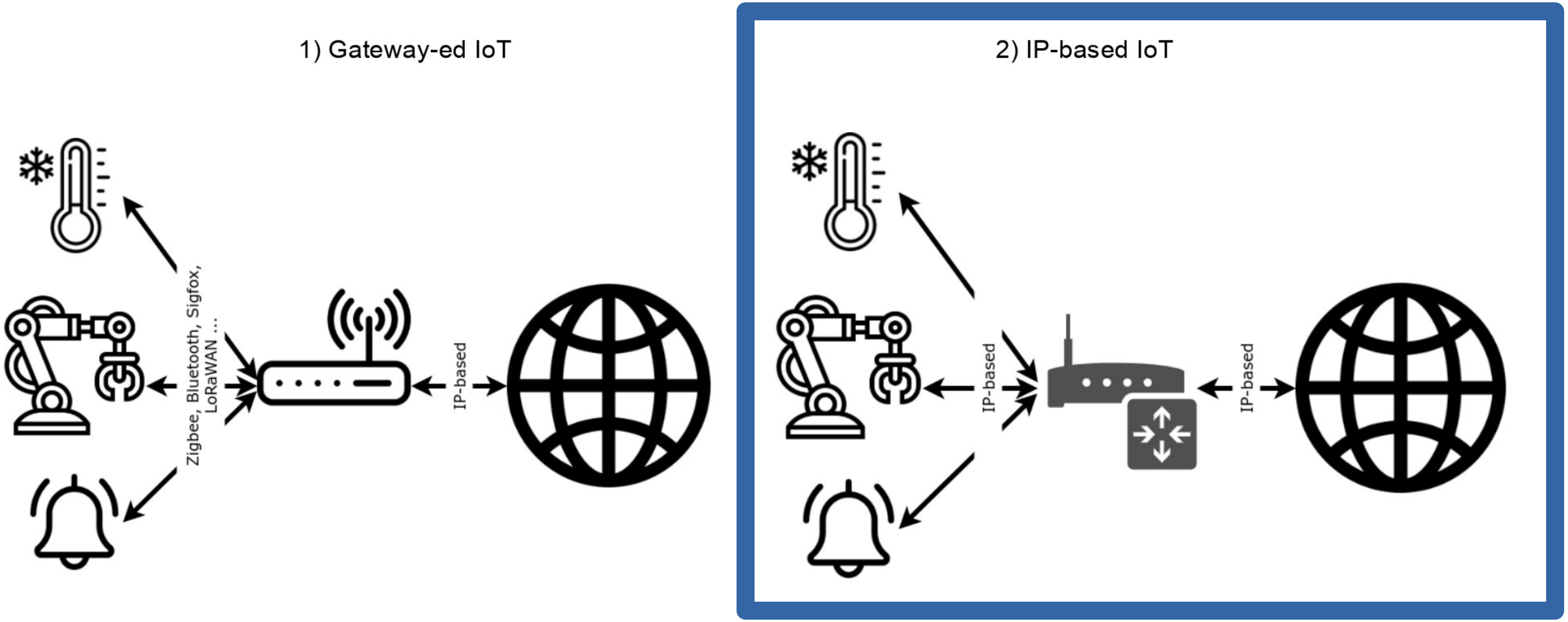
Development Environment

- We are not bound to a specific environment.
- Recommendation: use Visual Studio Code as an IDE/Editor
- What you will need:
  - A git repository where you can use to develop and use to send us the deliverables
  - Something to write, render/preview markdown files
  - Something to write JSON files and quickly validate them
  - Something to write Node.js code
  - Web Browser (Chrome-based or Firefox is recommended. We cannot evaluate other browsers)
  - A REST/HTTP client. Easy-to-use clients for MQTT and CoAP are also recommended

Recap

Two types of IoT

IoT Remote Lab



Course Contents

IoT Introduction	Architectures	Node.js	HTTP/CoAP	Sensors and Actuators
IoT Remote Lab	Web of Things	REST	MQTT/AMQP	Serial Protocols
Payload Formats	Thing Description	Scripting API / node-wot	Consumer Applications	Thing Applications

Why payloads now?

- We want to give you practical exercises to do.
- We can explain payloads after the Web of Things, but you would be listening all the time with nothing practical to do!

What is a payload?

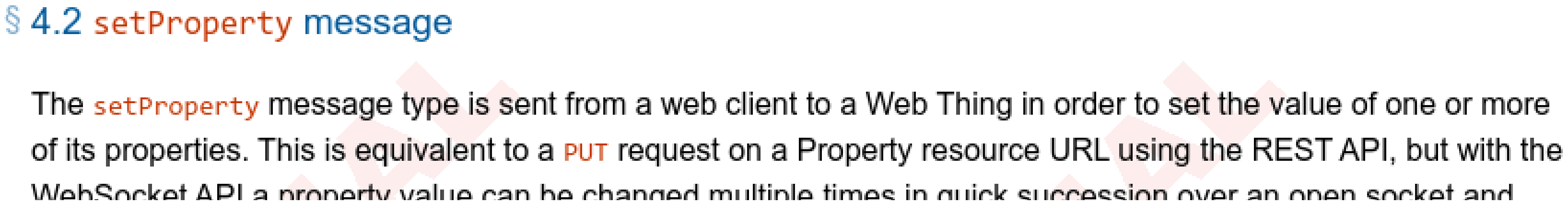
- In one sentence:

  Payload is the part of a message that has nothing to do with the protocol.
- Sometimes called the body of the message
- It is not mandatory in most protocols.
- Examples:
  - If a car is a protocol, then the what you put in your trunk or the passengers is the payload
  - If you are sending a PDF file, whether it is via email, handing a USB stick, downloading from a Website, that PDF file is the payload
  - If a sensor sends a temperature value as an integer, that value is the payload if it is HTTP, Zigbee or USB.

What it should not be?

We will get into more details in the REST lecture but here is a small rabbit hole:

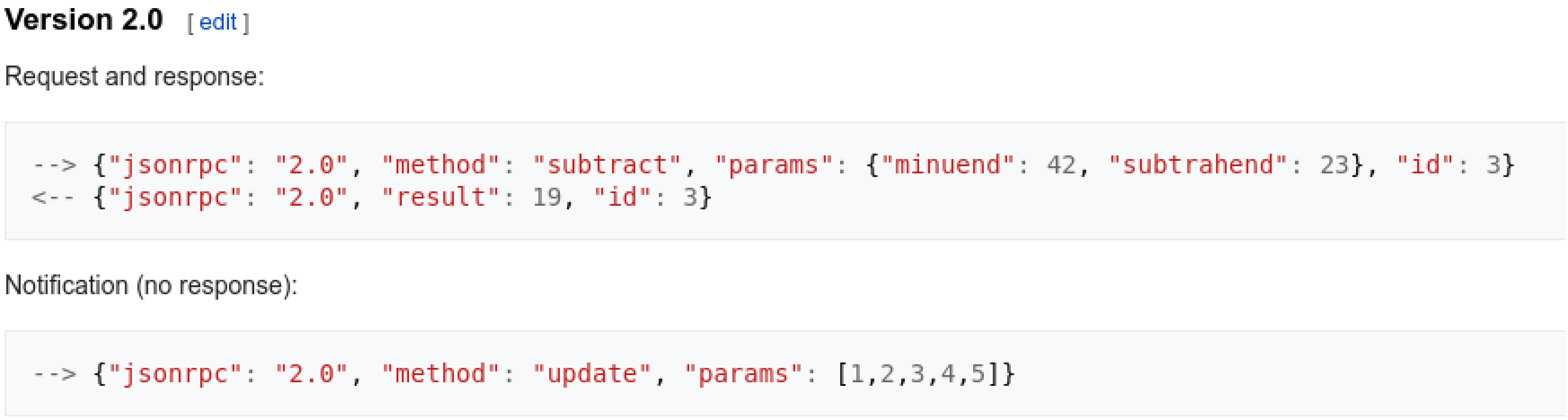
- It should not contain a protocol or a sub-protocol, i.e. payload should not tell what it should be done by itself!
  - Car example: The package destination should not be inside the luggage
  - Sending a F



§ 4.2 `setProperty` message

The `setProperty` message type is sent from a web client to a Web Thing in order to set the value of one or more of its properties. This is equivalent to a `PUT` request on a Property resource URL using the REST API, but with the WebSocket API a property value can be changed multiple times in quick succession over an open socket and

But isn't that obvi



Version 2.0 [edit]

Request and response:

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": {"minuend": 42, "subtrahend": 23}, "id": 3}
<-- {"jsonrpc": "2.0", "result": 19, "id": 3}
```

Notification (no response):

```
--> {"jsonrpc": "2.0", "method": "update", "params": [1,2,3,4,5]}
```

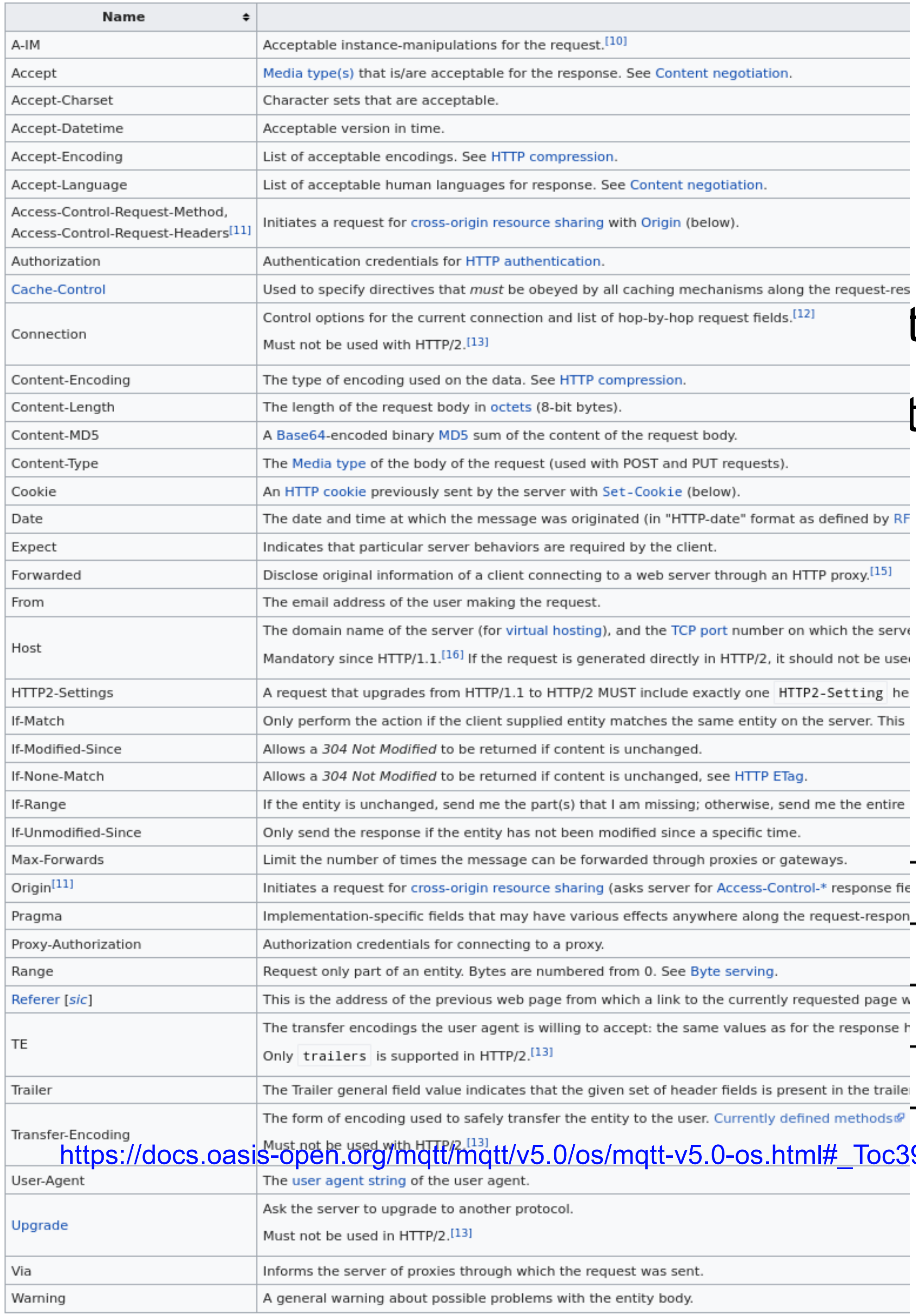
https://iot.mozilla.org/wot/#web-thing-websocket-api
https://en.wikipedia.org/wiki/JSON-RPC
https://martinfowler.com/articles/richardsonMaturityModel.html

What is a header?

- In one sentence:

  Header is part of a messa the protocol stack.

- In another way, whatever the header.
- Some examples from diffe



Name ⬍	
A-IM	Acceptable instance-manipulations for the request.[10]
Accept	Media type(s) that is/are acceptable for the response. See Content negotiation.
Accept-Charset	Character sets that are acceptable.
Accept-Datetime	Acceptable version in time.
Accept-Encoding	List of acceptable encodings. See HTTP compression.
Accept-Language	List of acceptable human languages for response. See Content negotiation.
Access-Control-Request-Method, Access-Control-Request-Headers[11]	Initiates a request for cross-origin resource sharing with Origin (below).
Authorization	Authentication credentials for HTTP authentication.
Cache-Control	Used to specify directives that *must* be obeyed by all caching mechanisms along the request-res
Connection	Control options for the current connection and list of hop-by-hop request fields.[12] Must not be used with HTTP/2.[13]
Content-Encoding	The type of encoding used on the data. See HTTP compression.
Content-Length	The length of the request body in octets (8-bit bytes).
Content-MD5	A Base64-encoded binary MD5 sum of the content of the request body.
Content-Type	The Media type of the body of the request (used with POST and PUT requests).
Cookie	An HTTP cookie previously sent by the server with `Set-Cookie` (below).
Date	The date and time at which the message was originated (in "HTTP-date" format as defined by RF
Expect	Indicates that particular server behaviors are required by the client.
Forwarded	Disclose original information of a client connecting to a web server through an HTTP proxy.[15]
From	The email address of the user making the request.
Host	The domain name of the server (for virtual hosting), and the TCP port number on which the serve Mandatory since HTTP/1.1.[16] If the request is generated directly in HTTP/2, it should not be use
HTTP2-Settings	A request that upgrades from HTTP/1.1 to HTTP/2 MUST include exactly one `HTTP2-Setting` he
If-Match	Only perform the action if the client supplied entity matches the same entity on the server. This
If-Modified-Since	Allows a *304 Not Modified* to be returned if content is unchanged.
If-None-Match	Allows a *304 Not Modified* to be returned if content is unchanged, see HTTP ETag.
If-Range	If the entity is unchanged, send me the part(s) that I am missing; otherwise, send me the entire
If-Unmodified-Since	Only send the response if the entity has not been modified since a specific time.
Max-Forwards	Limit the number of times the message can be forwarded through proxies or gateways.
Origin[11]	Initiates a request for cross-origin resource sharing (asks server for Access-Control-* response fie
Pragma	Implementation-specific fields that may have various effects anywhere along the request-respon
Proxy-Authorization	Authorization credentials for connecting to a proxy.
Range	Request only part of an entity. Bytes are numbered from 0. See Byte serving.
Referer [*sic*]	This is the address of the previous web page from which a link to the currently requested page w
TE	The transfer encodings the user agent is willing to accept: the same values as for the response h Only `trailers` is supported in HTTP/2.[13]
Trailer	The Trailer general field value indicates that the given set of header fields is present in the traile
Transfer-Encoding	The form of encoding used to safely transfer the entity to the user. Currently defined methods Must not be used with HTTP/2.[13]
User-Agent	The user agent string of the user agent.
Upgrade	Ask the server to upgrade to another protocol. Must not be used in HTTP/2.[13]
Via	Informs the server of proxies through which the request was sent.
Warning	A general warning about possible problems with the entity body.

3.3.1 PUBLISH Fixed Header

Figure 3-8 – PUBLISH packet Fixed Header

Bit	7		2	1	0
byte 1		M	QoS level		RETAIN
	0		X	X	X
byte 2…					

https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html#_Toc3901101

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

Headers now?

No :)

We will go into more detail in the content of the 2nd deliverable

Different Payload Types

- JSON (application/json)
- CBOR (application/cbor)
- XML (application/xml)
- Text (text/plain)
- Audio, video and much, much more: https://www.iana.org/assignments/media-types/media-types.xhtml

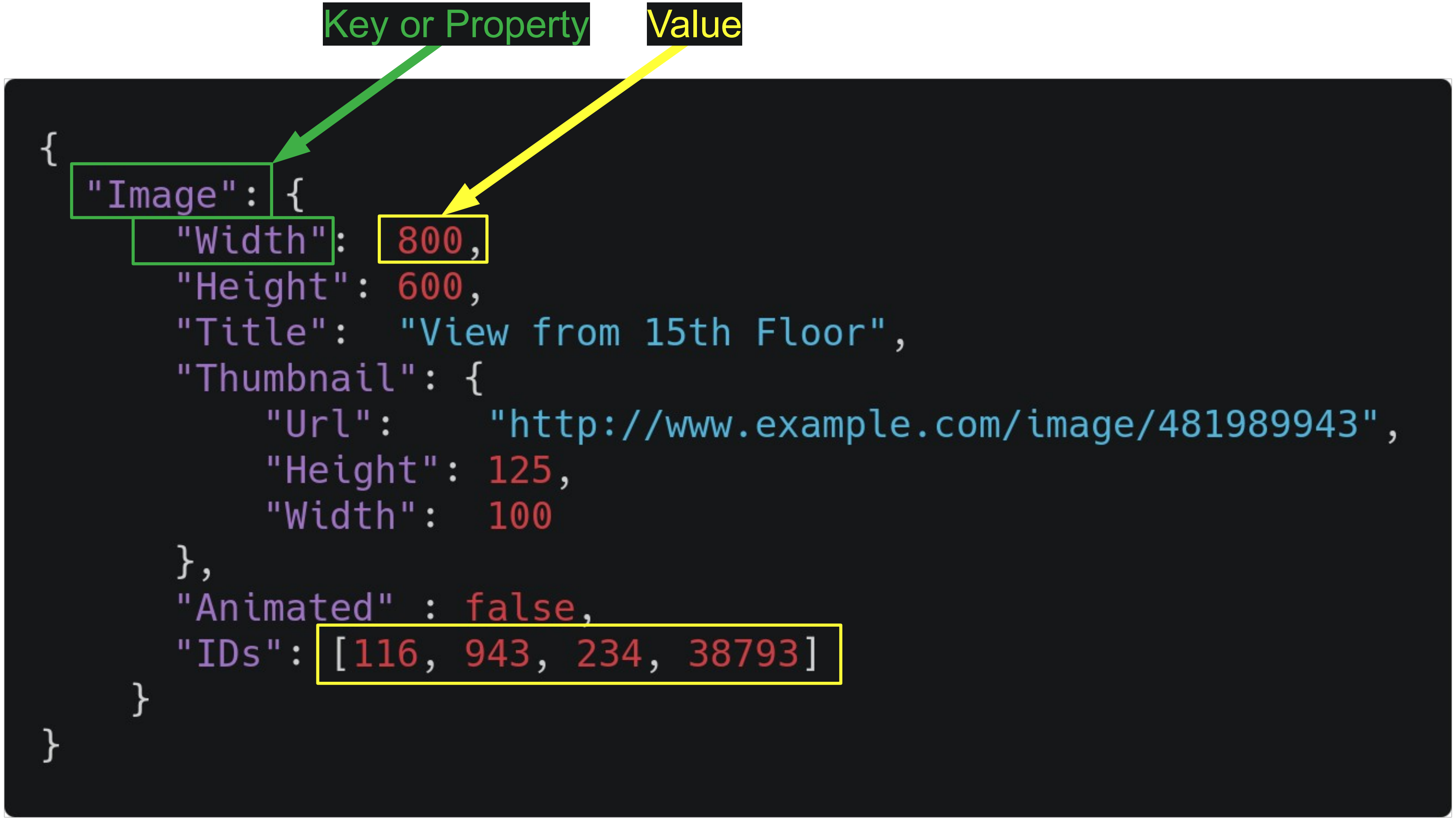
?

JSON

- Quick facts:
  - Standardized by ECMA and IETF in around 2013, current version is of 2017
  - Human readability was an important design choice
  - Used for data exchange but also for configuration files (like YAML)
  - Can be translated to other formats like XML, YAML etc.

JSON Examples

JSON Object



Key or Property

Value

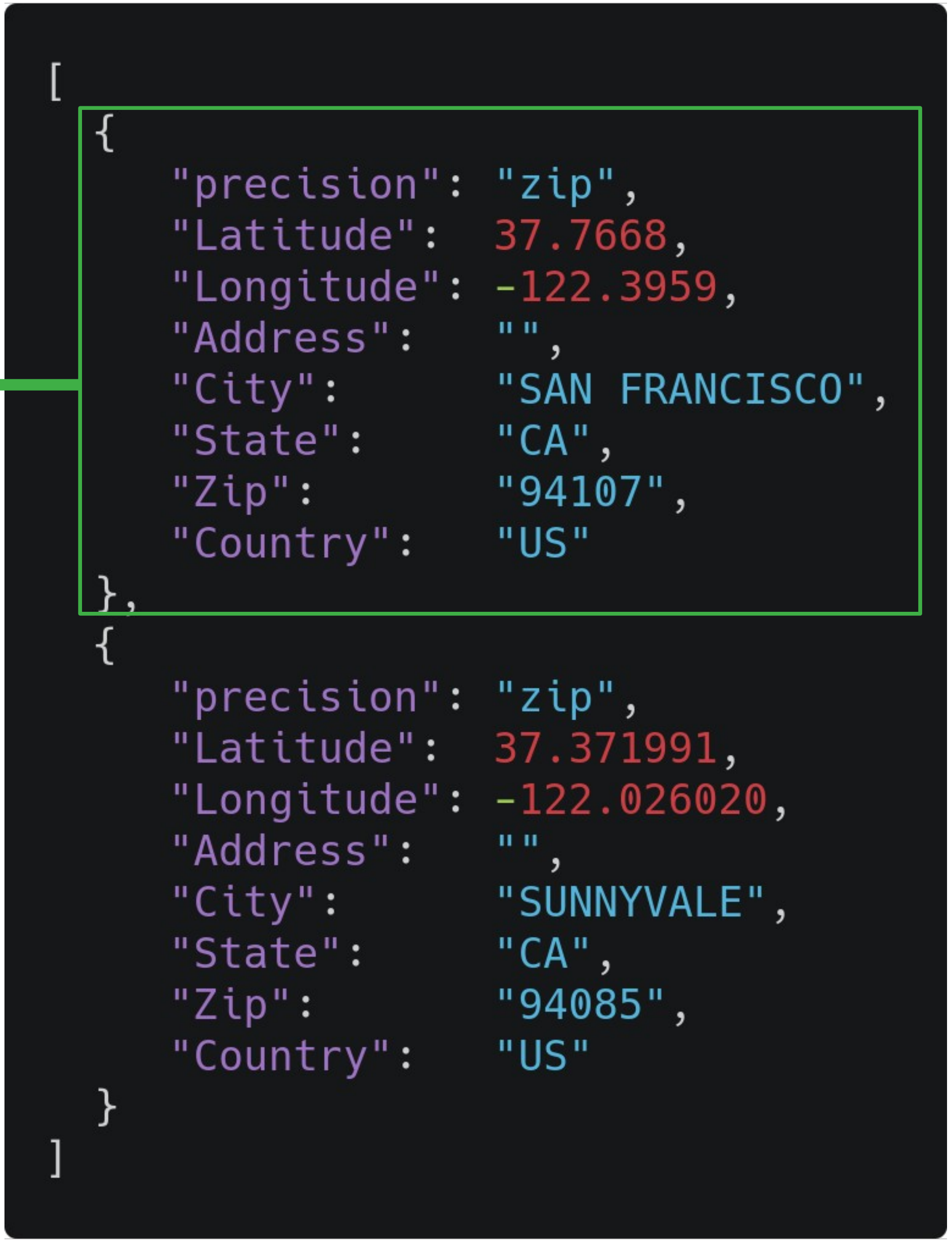
```
{
  "Image": {
      "Width":  800,
      "Height": 600,
      "Title":  "View from 15th Floor",
      "Thumbnail": {
          "Url":    "http://www.example.com/image/481989943",
          "Height": 125,
          "Width":  100
      },
      "Animated" : false,
      "IDs": [116, 943, 234, 38793]
    }
}
```

https://tools.ietf.org/html/rfc8259

JSON Examples

JSON Array

item

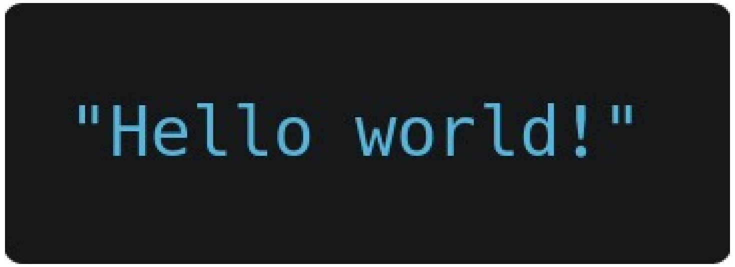


```
[
  {
     "precision": "zip",
     "Latitude":  37.7668,
     "Longitude": -122.3959,
     "Address":   "",
     "City":      "SAN FRANCISCO",
     "State":     "CA",
     "Zip":       "94107",
     "Country":   "US"
  },
  {
     "precision": "zip",
     "Latitude":  37.371991,
     "Longitude": -122.026020,
     "Address":   "",
     "City":      "SUNNYVALE",
     "State":     "CA",
     "Zip":       "94085",
     "Country":   "US"
  }
]
```

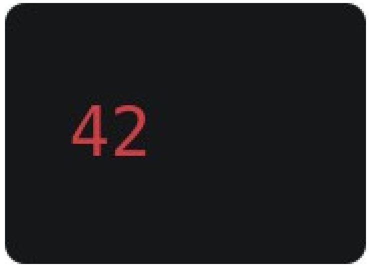
https://tools.ietf.org/html/rfc8259

JSON Examples

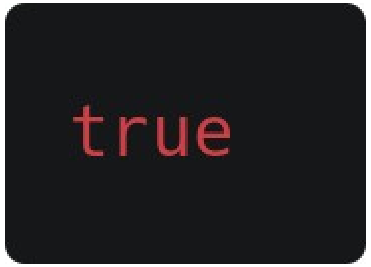
Primitive Types



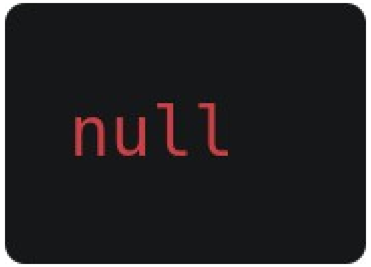
```
"Hello world!"
```



```
42
```



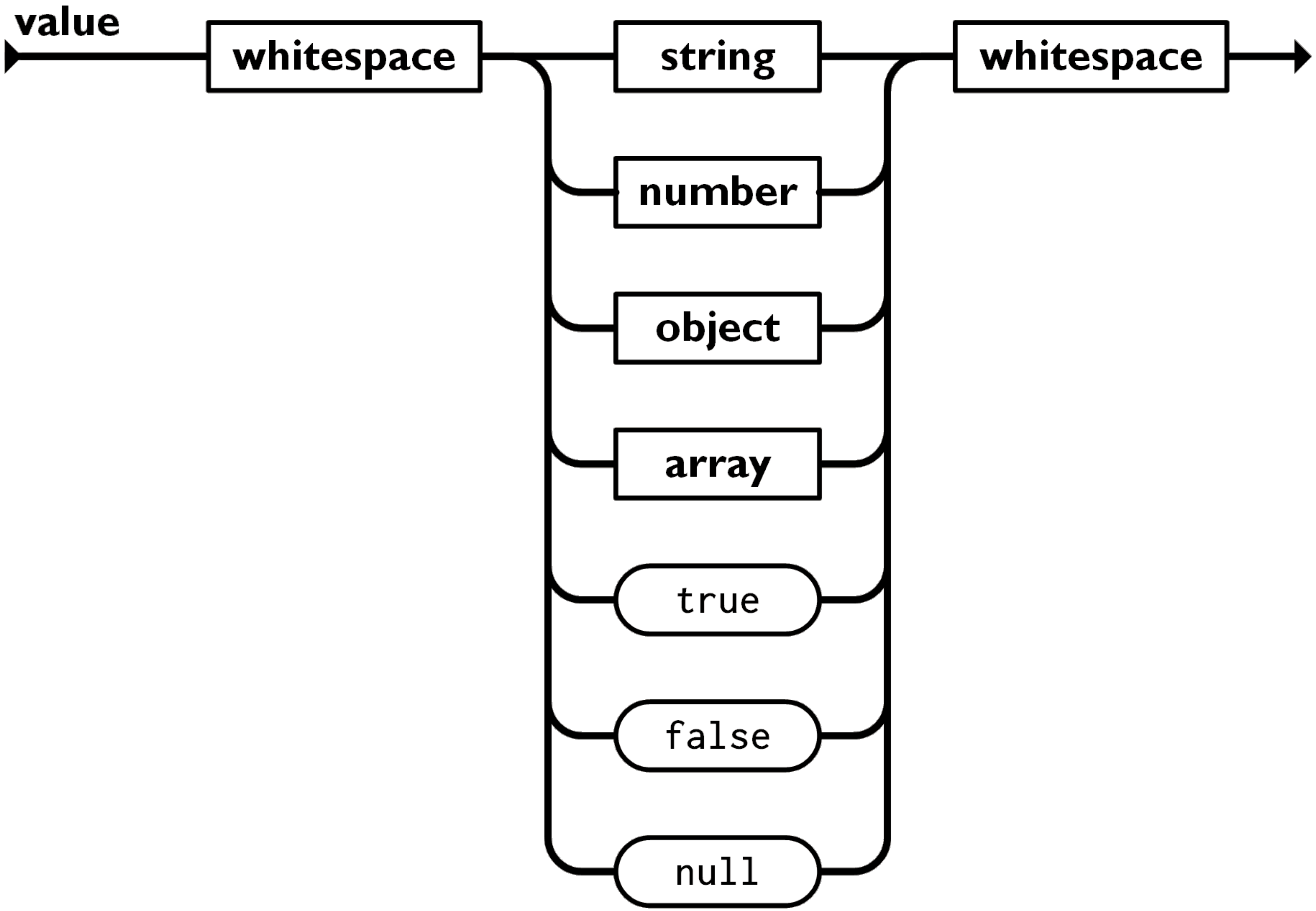
```
true
```



```
null
```

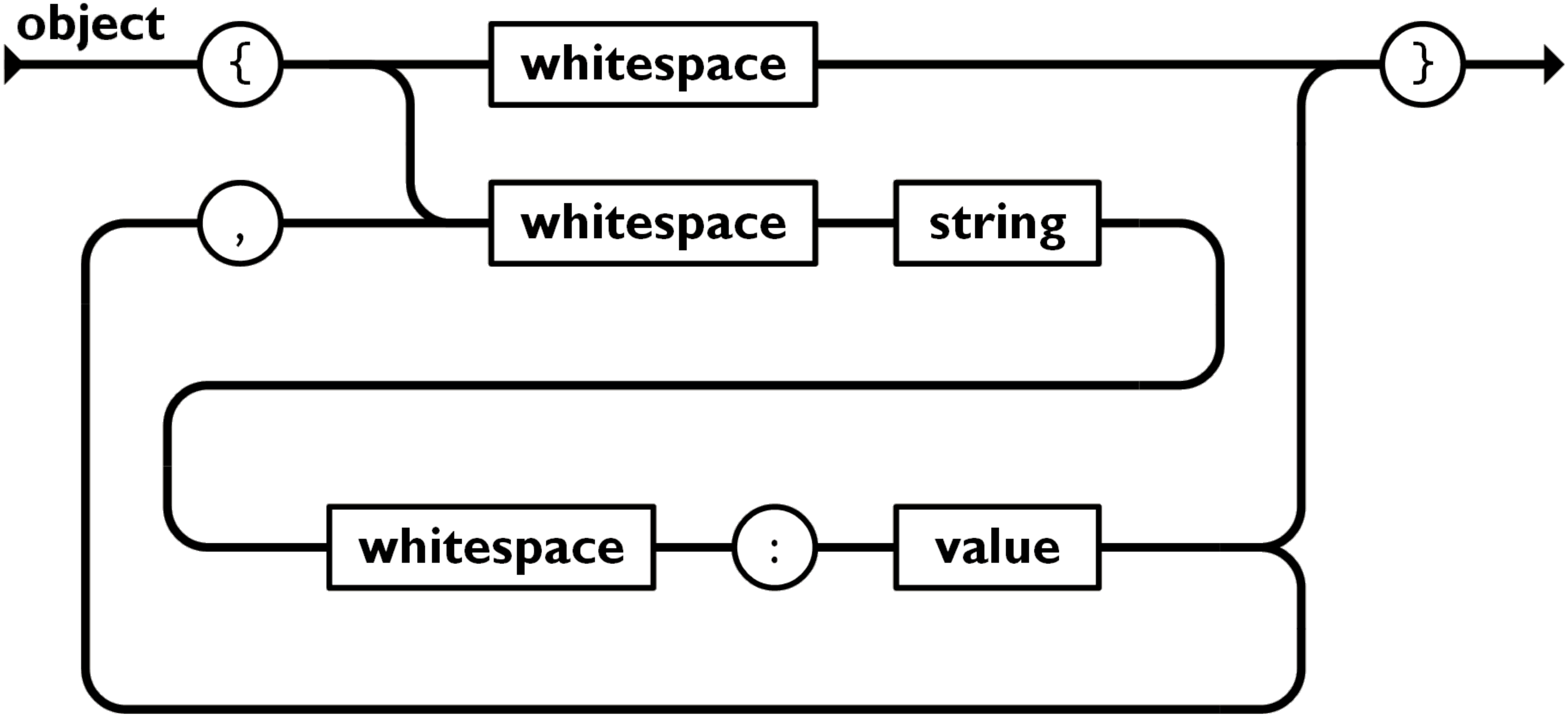
https://tools.ietf.org/html/rfc8259

JSON Structure



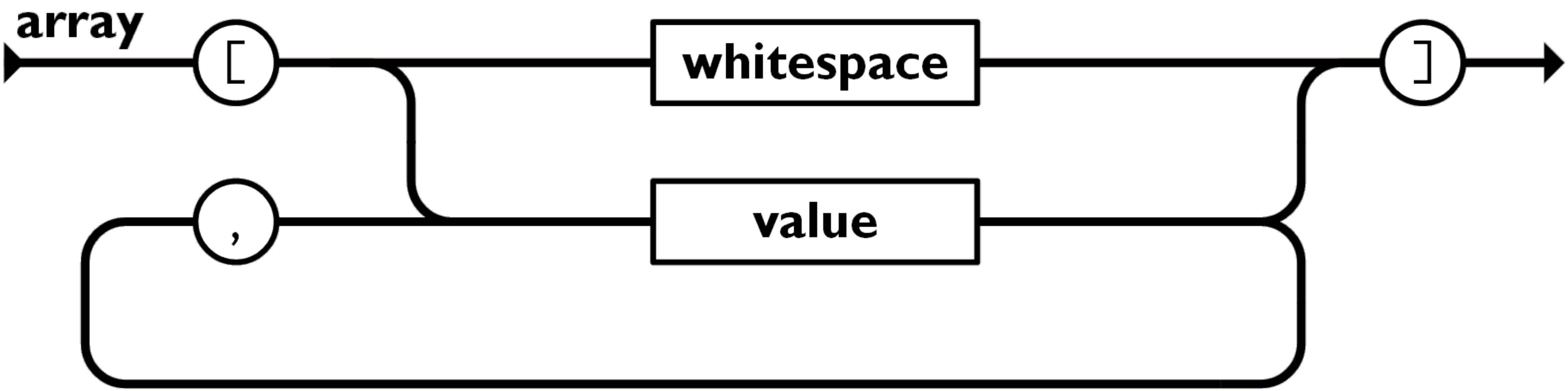
https://www.json.org/json-en.html

JSON Structure



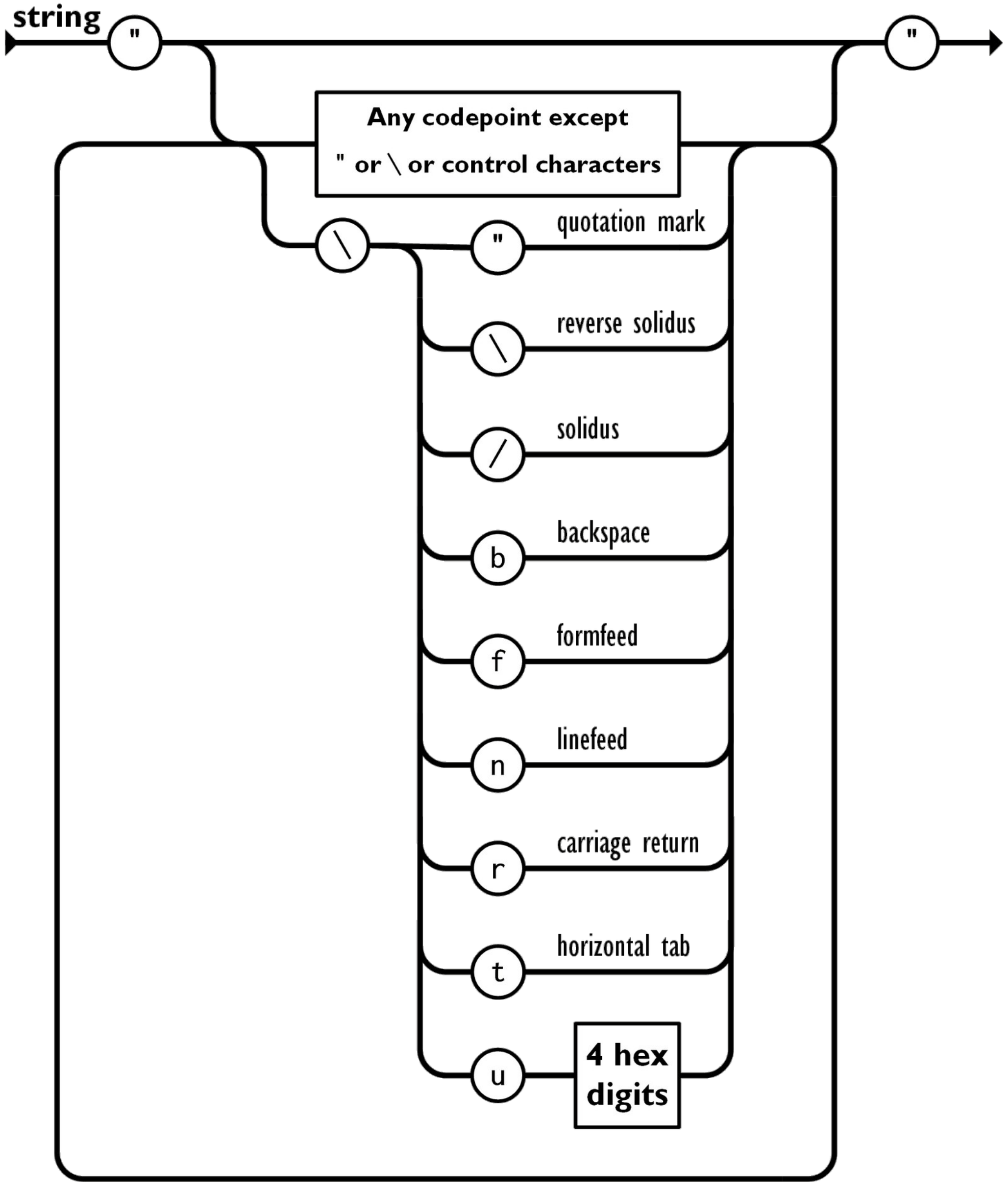
https://www.json.org/json-en.html

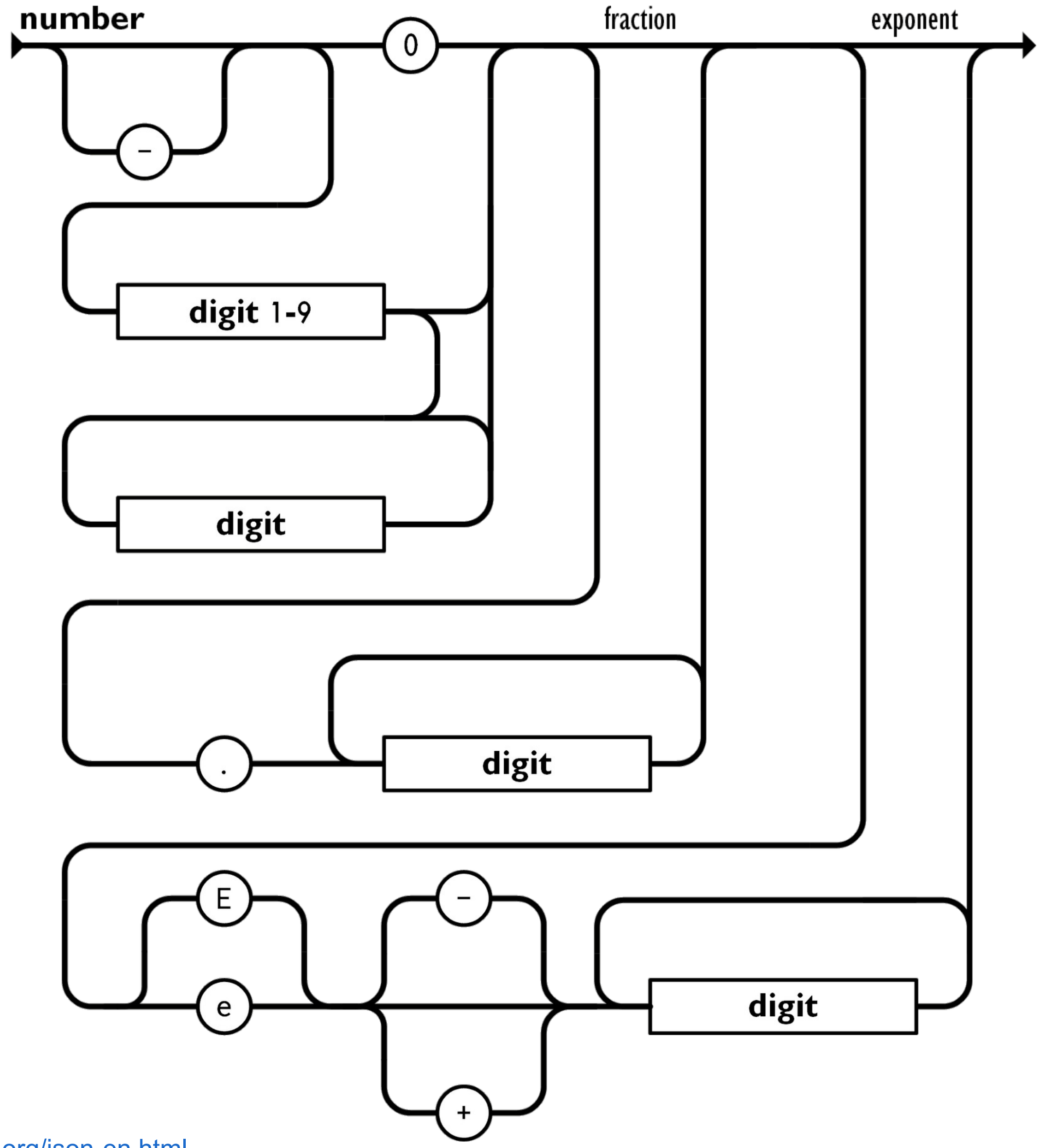
JSON Structure



https://www.json.org/json-en.html

JSON Structure





https://www.json.org/json-en.html

Why is an array called array?

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- Kind of confusing?
  - The super early versions of JSON was only for objects and would not allow the primitive types to exist on their own.
  - Since JavaScript was used for Web-based front ends and needed to exchange data with the servers/backends, its way of structuring data got popular. Now, pretty much no one uses XML for the communication between Web server and clients.

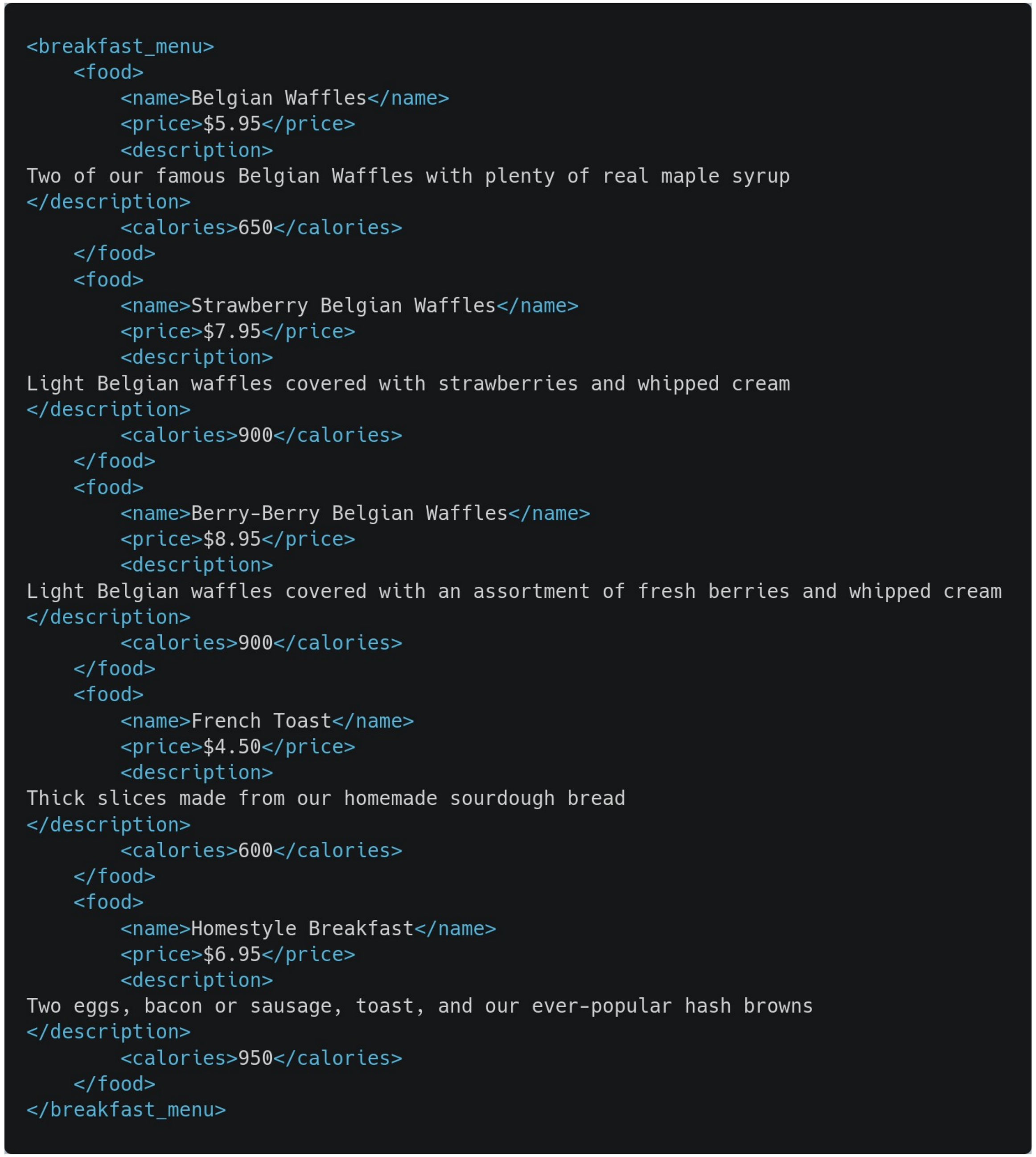
JSON	Python
object	dict
array	list
string	str
number	int,long,float
true, false	True, False
null	None

Working with JSON in your Editor

- Some little things that will help you:
  - Beautifying JSON files
  - (Auto)Validating JSON
  - Expand and compact different levels of JSON
  - Highlighting

Other formats

XML

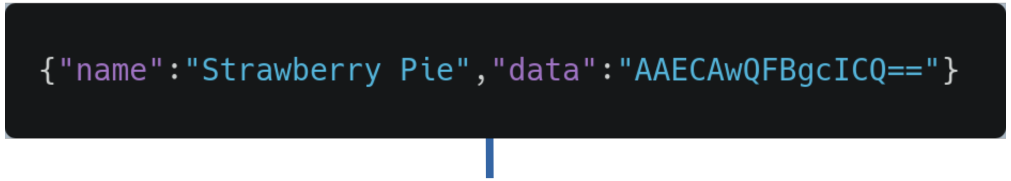


```
<breakfast_menu>
    <food>
        <name>Belgian Waffles</name>
        <price>$5.95</price>
        <description>
Two of our famous Belgian Waffles with plenty of real maple syrup
</description>
        <calories>650</calories>
    </food>
    <food>
        <name>Strawberry Belgian Waffles</name>
        <price>$7.95</price>
        <description>
Light Belgian waffles covered with strawberries and whipped cream
</description>
        <calories>900</calories>
    </food>
    <food>
        <name>Berry-Berry Belgian Waffles</name>
        <price>$8.95</price>
        <description>
Light Belgian waffles covered with an assortment of fresh berries and whipped cream
</description>
        <calories>900</calories>
    </food>
    <food>
        <name>French Toast</name>
        <price>$4.50</price>
        <description>
Thick slices made from our homemade sourdough bread
</description>
        <calories>600</calories>
    </food>
    <food>
        <name>Homestyle Breakfast</name>
        <price>$6.95</price>
        <description>
Two eggs, bacon or sausage, toast, and our ever-popular hash browns
</description>
        <calories>950</calories>
    </food>
</breakfast_menu>
```

https://www.w3schools.com/xml/simple.xml

Other formats

CBOR



```
{"name":"Strawberry Pie","data":"AAECAwQFBgcICQ=="}
```

Encode to CBOR

a2646e616d656e5374726177626572727920506965696a7065675f646174614a00010203040506070809

What it means

```
a2                  -- Map, 2 pairs
  64                -- String, length: 4
    6e616d65        -- {Key:0}, "name"
  6e                -- String, length: 14
    5374726177626572727920506965 -- {Val:0}, "Strawberry Pie"
  64                -- String, length: 4
    64617461 -- {Key:1}, "data"
  4a                -- Bytes, length: 10
    00010203040506070809 -- {Val:1}, 00010203040506070809
```

https://www.endpoint.com/blog/2019/03/18/extensible-binary-encoding-with-cbor

Modeling Payloads

- Many payload formats offer a way to describe what a payload instance should look like
- Generally, these are called Schema Languages.
- Examples:
  - JSON Schema
  - XML Schema
  - RDF Schema

  etc.

Modeling Payloads

- The main idea is to enable senders (e.g. clients) to understand how the request should look and for the receiver to automatically validate the payloads.
- In the receiver end, you can thus do:

```
bool isValid = MyValidator.validate(schema, payload)
```

JSON Schema

- (Almost) a standard to define/model/describe JSON Payloads
  - Does not have RFC status, 8th draft
- Homepage at http://json-schema.org/
- Standard at https://tools.ietf.org/html/draft-handrews-json-schema-validation-02#section-6.1
- A more friendly version that is also tutorial is at https://json-schema.org/understanding-json-schema/index.html

JSON Schema

- Although, it is a JSON on its own, it is **metadata**
- Some basic JSON Schemas:

JSON	
object	{ "type": "object" }
array	{ "type": "array" }
string	{ "type": "string" }
number	{ "type": "number" } { "type": "integer" }
true, false	{ "type": "boolean" }
null	{ "type": "null" }

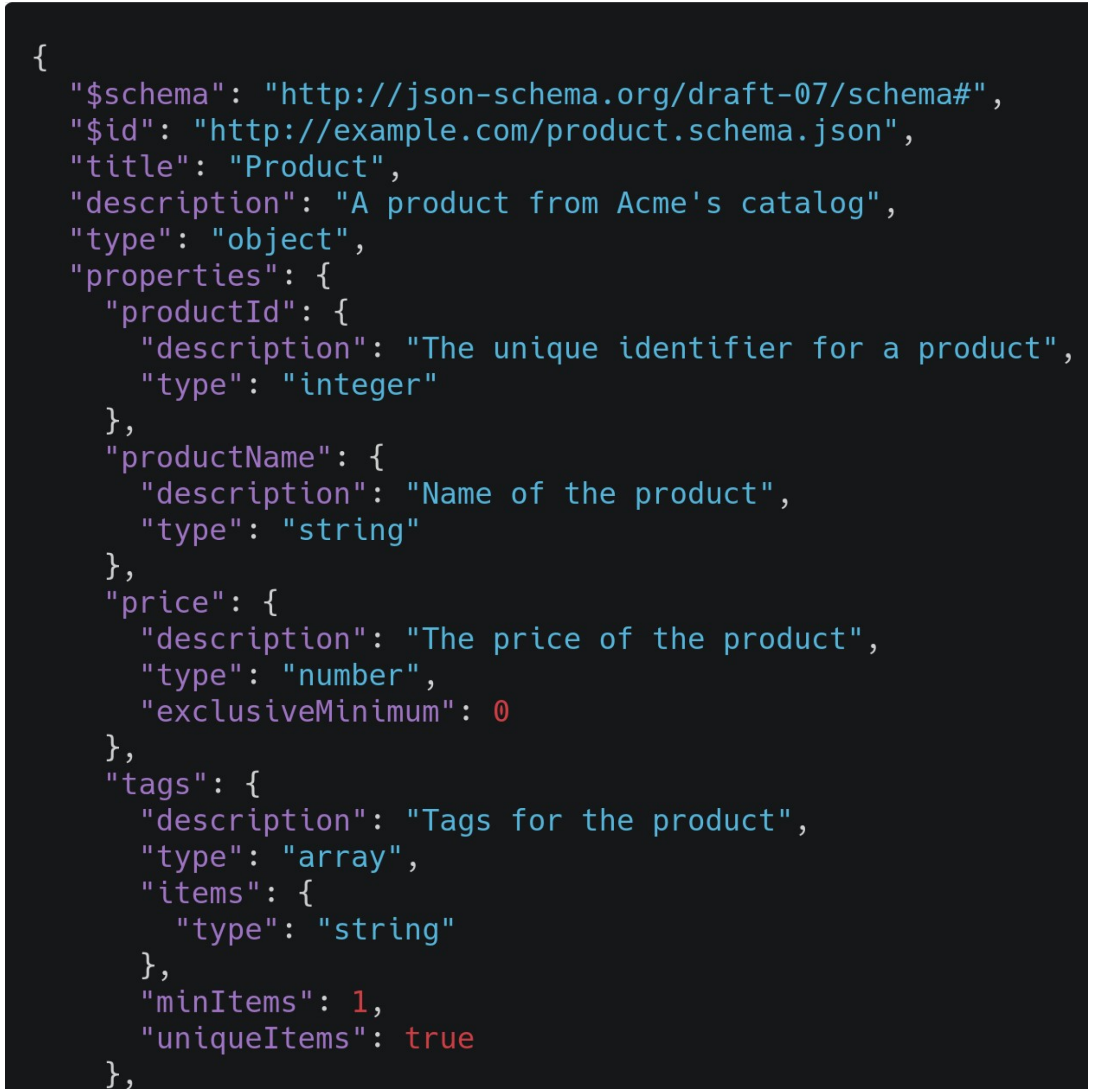
JSON Schema

- Of course, this is not all! We want to:
  - specify the length of a **string**, define regular expressions, use well-known formats such as URIs, email addresses, IP addresses etc.
  - specify the minimum, maximum for **numbers**, whether they are multiples of a base. This allows to describe *float*, *long* kind of types
  - tell how many items are allowed for **arrays**, what should be the type of values (unordered and ordered), whether there can be unspecified items, whether they should be unique
  - for **objects**, specify the types of properties, required properties, whether there can be unspecified properties, amount of properties, how should property names look

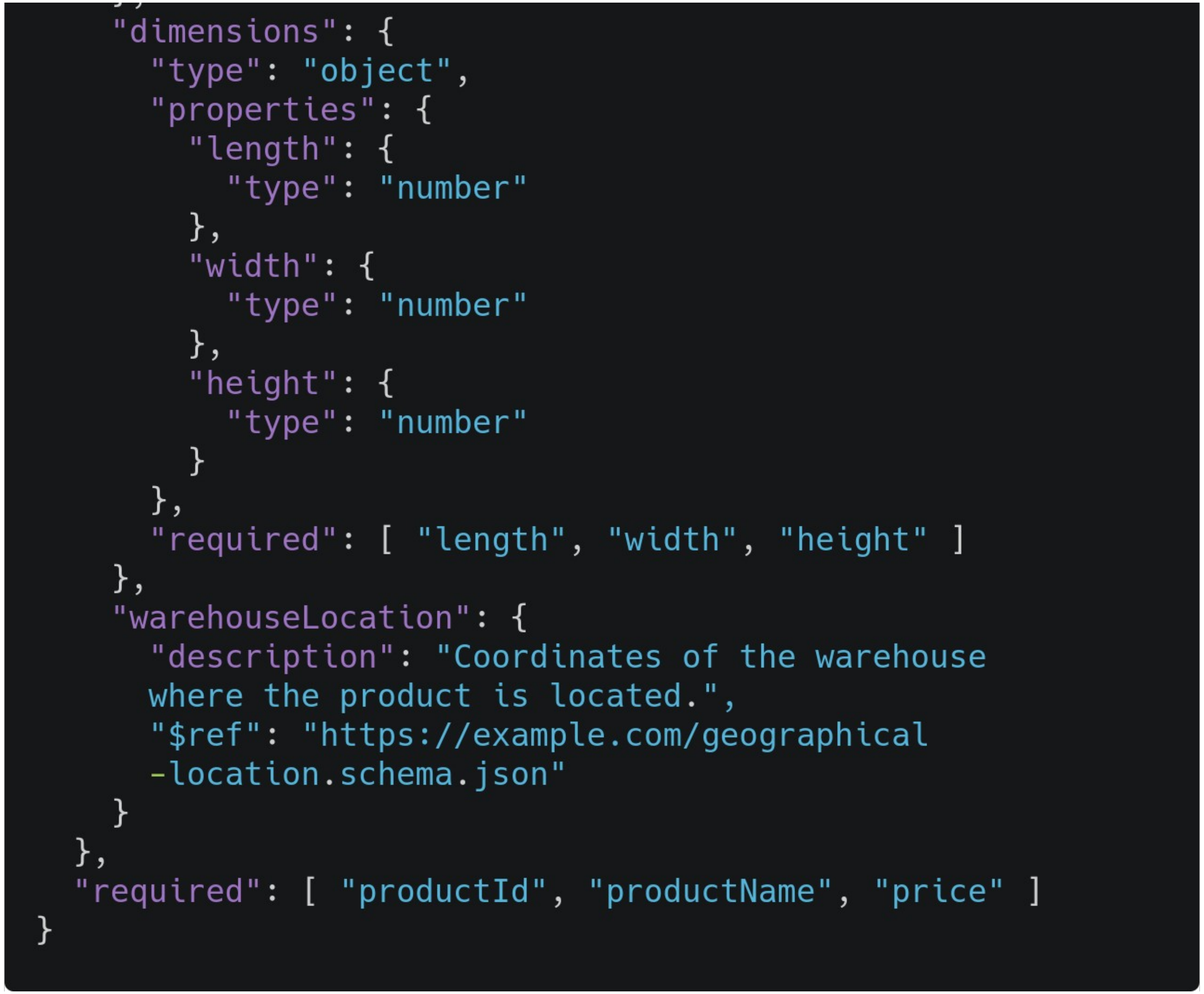
JSON Schema

- In the next slides, you will see some detailed JSON Schemas.
- The best way to learn is to practice! There are many online tools to write a JSON Schema and validate different JSON payloads accordingly.
  - Pay attention to the Draft version! Use tools with Draft V6 and up
  - Some have a downloadable library or tell what library they use
  - Suggestions:
    - https://jsonschemalint.com uses ajv for validation (Node.js)
    - https://www.jsonschemavalidator.net/ uses its own validator (.NET)

JSON Schema Examples



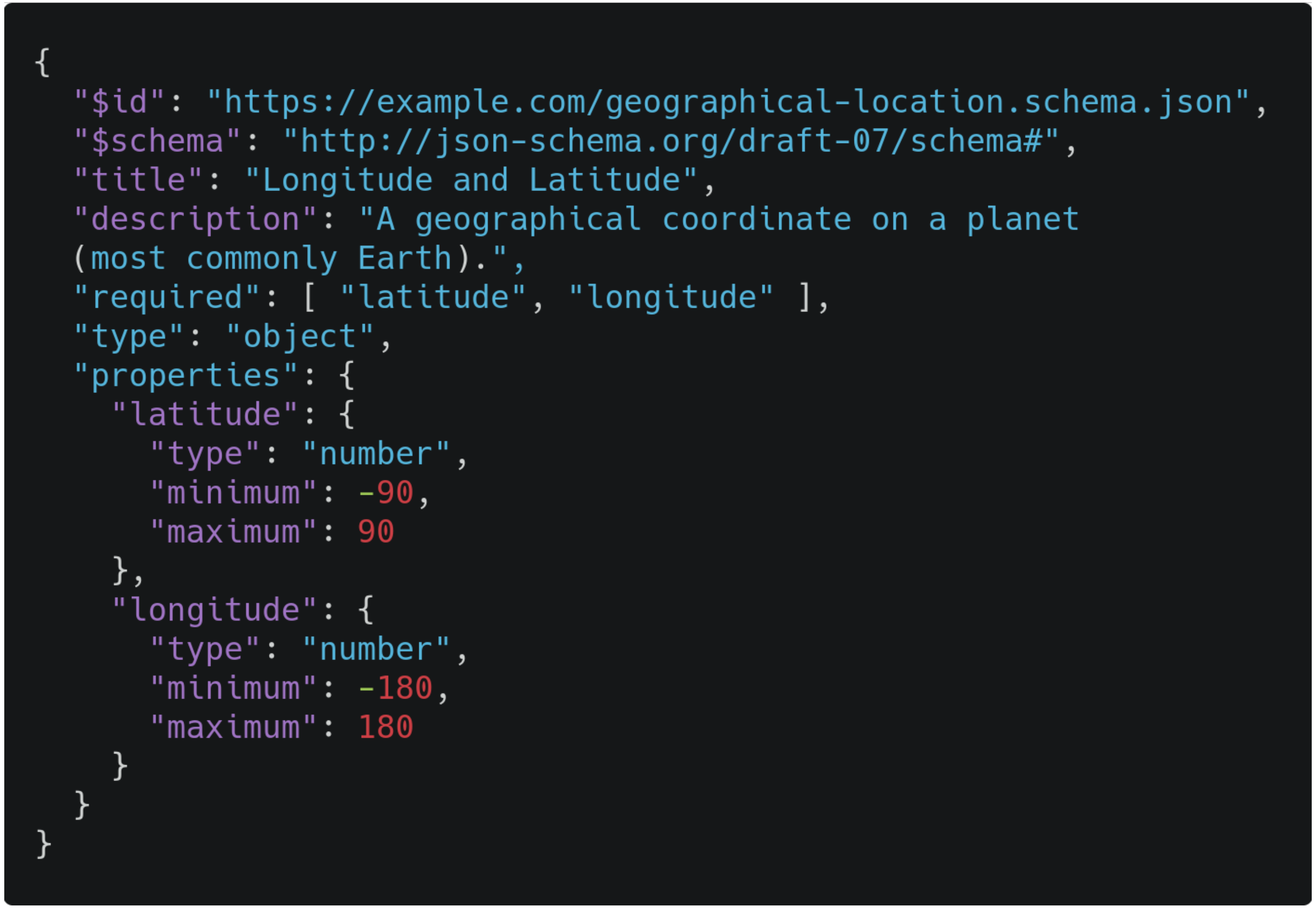
```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://example.com/product.schema.json",
  "title": "Product",
  "description": "A product from Acme's catalog",
  "type": "object",
  "properties": {
    "productId": {
      "description": "The unique identifier for a product",
      "type": "integer"
    },
    "productName": {
      "description": "Name of the product",
      "type": "string"
    },
    "price": {
      "description": "The price of the product",
      "type": "number",
      "exclusiveMinimum": 0
    },
    "tags": {
      "description": "Tags for the product",
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
```



```
    },
    "dimensions": {
      "type": "object",
      "properties": {
        "length": {
          "type": "number"
        },
        "width": {
          "type": "number"
        },
        "height": {
          "type": "number"
        }
      },
      "required": [ "length", "width", "height" ]
    },
    "warehouseLocation": {
      "description": "Coordinates of the warehouse
      where the product is located.",
      "$ref": "https://example.com/geographical
      -location.schema.json"
    }
  },
  "required": [ "productId", "productName", "price" ]
}
```

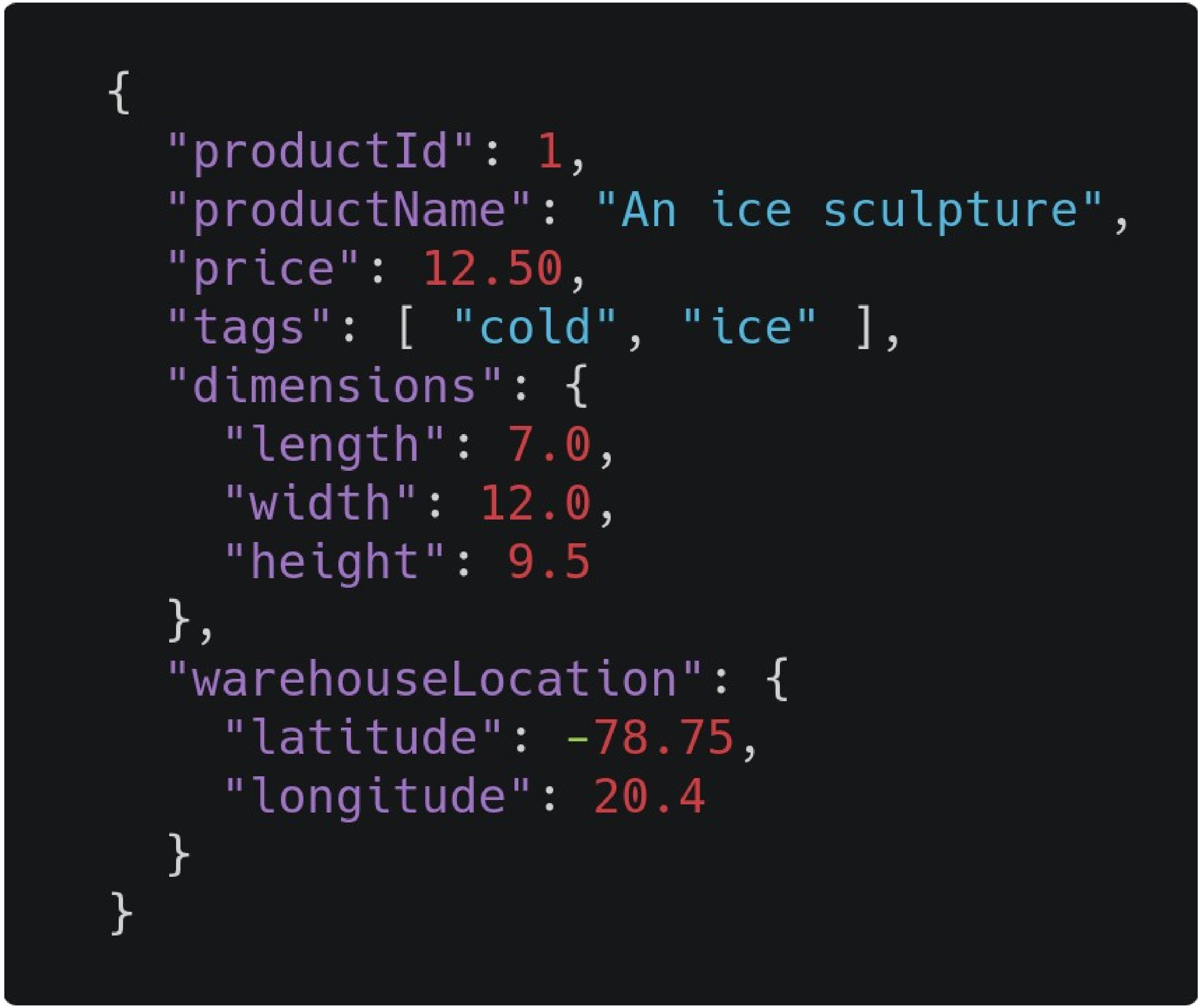
http://json-schema.org/learn/getting-started-step-by-step.html#properties-deeper

Previous Example Continued



```
{
  "$id": "https://example.com/geographical-location.schema.json",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Longitude and Latitude",
  "description": "A geographical coordinate on a planet
  (most commonly Earth).",
  "required": [ "latitude", "longitude" ],
  "type": "object",
  "properties": {
    "latitude": {
      "type": "number",
      "minimum": -90,
      "maximum": 90
    },
    "longitude": {
      "type": "number",
      "minimum": -180,
      "maximum": 180
    }
  }
}
```

Previous Example Continued



```
{
  "productId": 1,
  "productName": "An ice sculpture",
  "price": 12.50,
  "tags": [ "cold", "ice" ],
  "dimensions": {
    "length": 7.0,
    "width": 12.0,
    "height": 9.5
  },
  "warehouseLocation": {
    "latitude": -78.75,
    "longitude": 20.4
  }
}
```

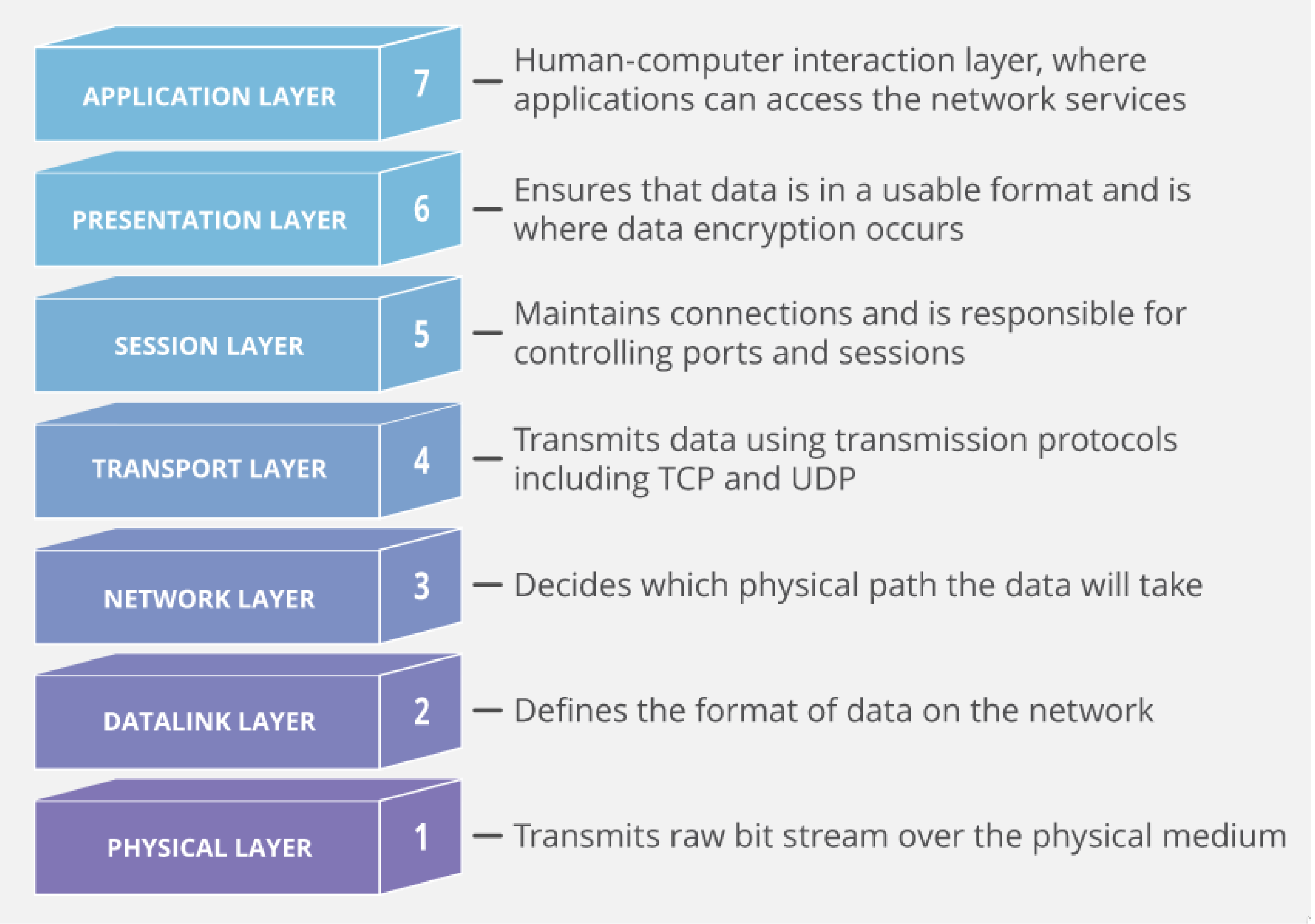
Going Further

- We do not have time to go into every feature of JSON Schema, the rest is for self-learning!

A little break and then

NETWORKED DEVICE ARCHITECTURES

OSI Layers



APPLICATION LAYER 7 — Human-computer interaction layer, where applications can access the network services

PRESENTATION LAYER 6 — Ensures that data is in a usable format and is where data encryption occurs

SESSION LAYER 5 — Maintains connections and is responsible for controlling ports and sessions

TRANSPORT LAYER 4 — Transmits data using transmission protocols including TCP and UDP

NETWORK LAYER 3 — Decides which physical path the data will take

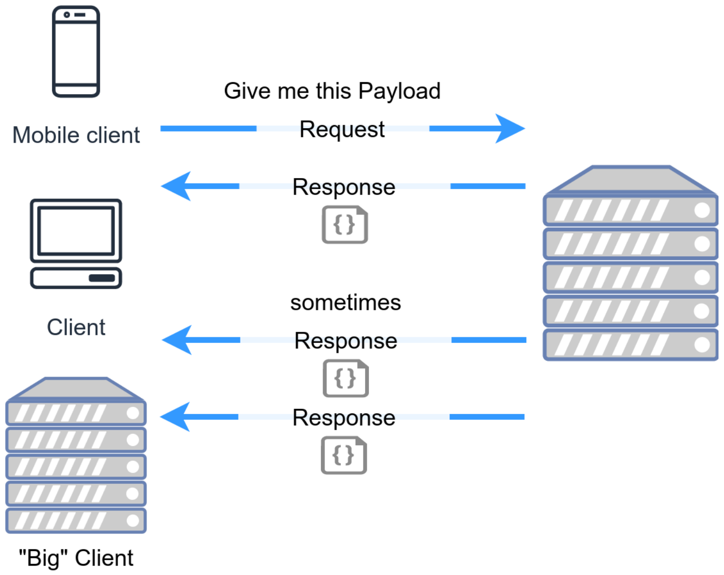
DATALINK LAYER 2 — Defines the format of data on the network

PHYSICAL LAYER 1 — Transmits raw bit stream over the physical medium

https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/

Server-Client

- A server waits for requests and responds to them.
- It does **not** imply request-response pattern, it is possible to do eventing.
- Is very similar to Master-Slave where the slave is the server and master is the client



Server-Client

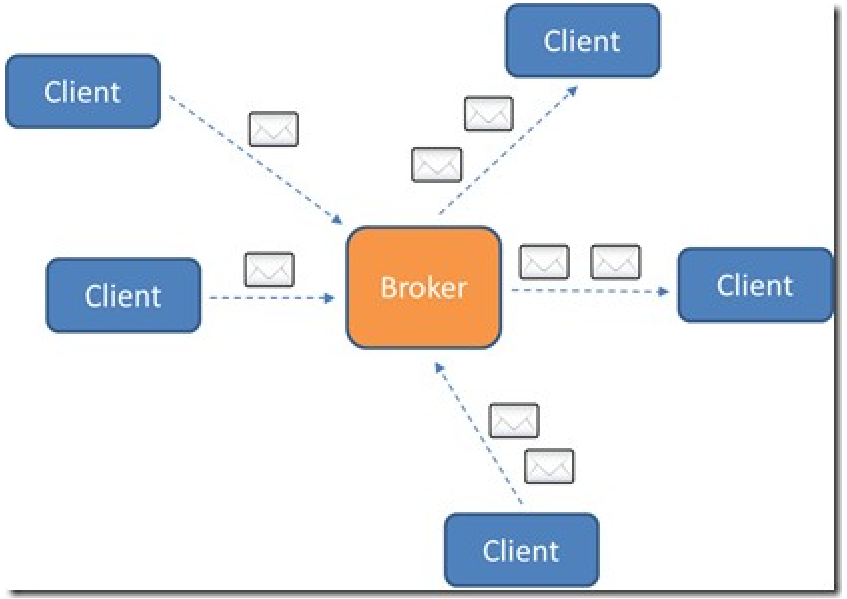
- Common protocols:
  - HTTP
  - Websockets
  - CoAP
  - IMAP
  - Modbus (Master-Slave)
  - OPC Data Access (from OPC-UA)
  - WHOIS

Server-Client

- Separation of concerns
  - One of the most important principles of REST architectures
  - Enables service-oriented architectures
- Server and Client at the same time!
  - Later on we will refer to this as Servient.
  - A single device or software can exhibit both behaviors. For example, reading temperature values from different server sensors and serve the average value for other clients.

Broker-Client(s)

- Also referred to as Publisher-Subscriber but Broker-Client is more specific since there can be PubSub without broker
- Multiple entities (clients) connect to a central entity (broker)
- Broker contains no application logic, i.e. you do not build an application or serve an application in the broker
- It does **not** imply push-based eventing/messaging, request-response pattern can be done.



http://www.embedded101.com/Develop-M2M-IoT-Devices-Ebook/DevelopM2MIoTDevicesContent/articleid/219?dnnprintmode=true&mid=948&SkinSrc=[G]Skins%2F_default%2FNo+Skin&ContainerSrc=[G]Containers%2F_default%2FNo+Container

Broker-Client(s)

- Example protocols:
  - MQTT
  - AMQP
  - Apache Kafka

Peer to Peer

- A „true“ distributed system where there is no single point to get the data from
- Requires the entities to discover each other
- Examples:
  - BitTorrent
  - Websocket (quite recent feature)
  - Bitcoin
  - Lower layer routing protocols (RPL)
    - Our SDIOT lecture covers this. Also see here: https://www.tex4tum.de/rpl

One part of deliverable 1 will be available in Moodle today, focusing on JSON and JSON Schema

Wrap-Up

- Payloads are application specific data that is sent over a protocol while having nothing to do with the protocol
- JSON is one payload type that we will use in this course
- JSON Schema allows to specify JSON payloads
- Multiple networked device architectures exist, we will focus on Server-Client and Broker-Client
- Watch Moodle for a part of deliverable 1