JSON & Third Party APIs

Week 12

What is JSON?

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a syntax for storing and exchanging data.
- JSON is an easier-to-use alternative to XML.
- JSON is language independent *
- JSON is "self-describing" and easy to understand

JSON vs XML

Json

```
{
   "employees": [
      {
         "firstName":"John",
         "lastName":"Doe"
      },
      {
         "firstName":"Anna",
         "lastName":"Smith"
      },
      {
         "firstName":"Peter",
         "lastName":"Jones"
      },
   ]
}
```

XML

```
<employees>
   <employee>
      <firstName>John</firstName>
      <lastName>Doe</lastName>
   </employee>
   <employee>
      <firstName>Anna</firstName>
      <lastName>Smith</lastName>
   </employee>
   <employee>
      <firstName>Peter</firstName>
      <lastName>Jones</lastName>
   </employee>
</employees>
```

JSON vs XML

Much Like XML Because

- Both JSON and XML are "self describing" (human readable)
- Both JSON and XML are hierarchical
- Both JSON and XML can be parsed and used by lots of programming languages
- Both JSON and XML can be fetched with an XMLHttpRequest

Much Unlike XML Because

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays
- JSON is a string

Why JSON?

For AJAX applications, JSON can be faster and easier than XML:

Using XML

- Fetch an XML document
- Use the XML DOM to loop through the document
- Extract values and store in variables

Using JSON

- Fetch a JSON string
- JSON.Parse the JSON string

JSON Syntax

- JSON syntax is derived from JavaScript object syntax:
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays

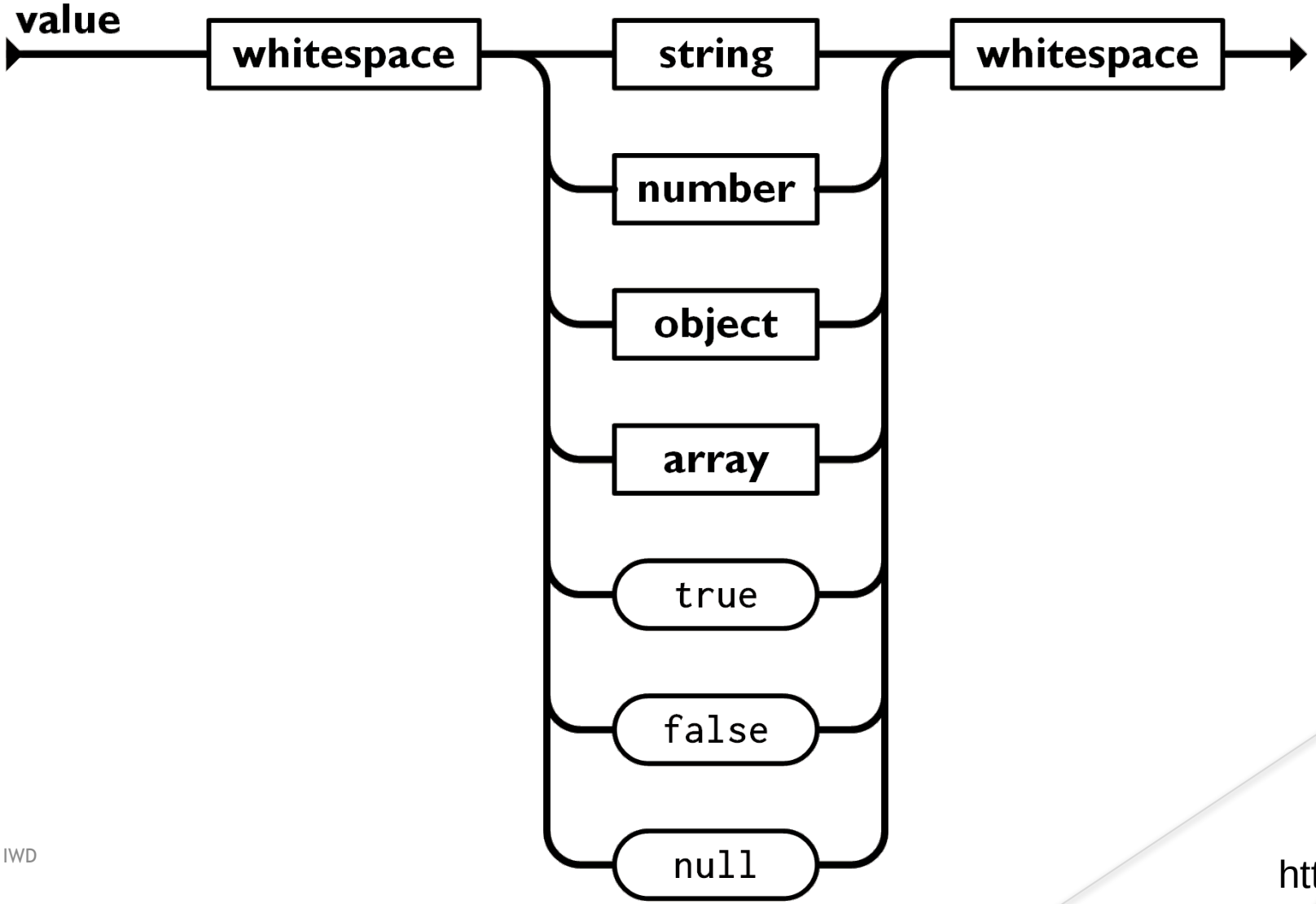
Except:

  - Data names are strings
  - Methods are not supported

JSON data types

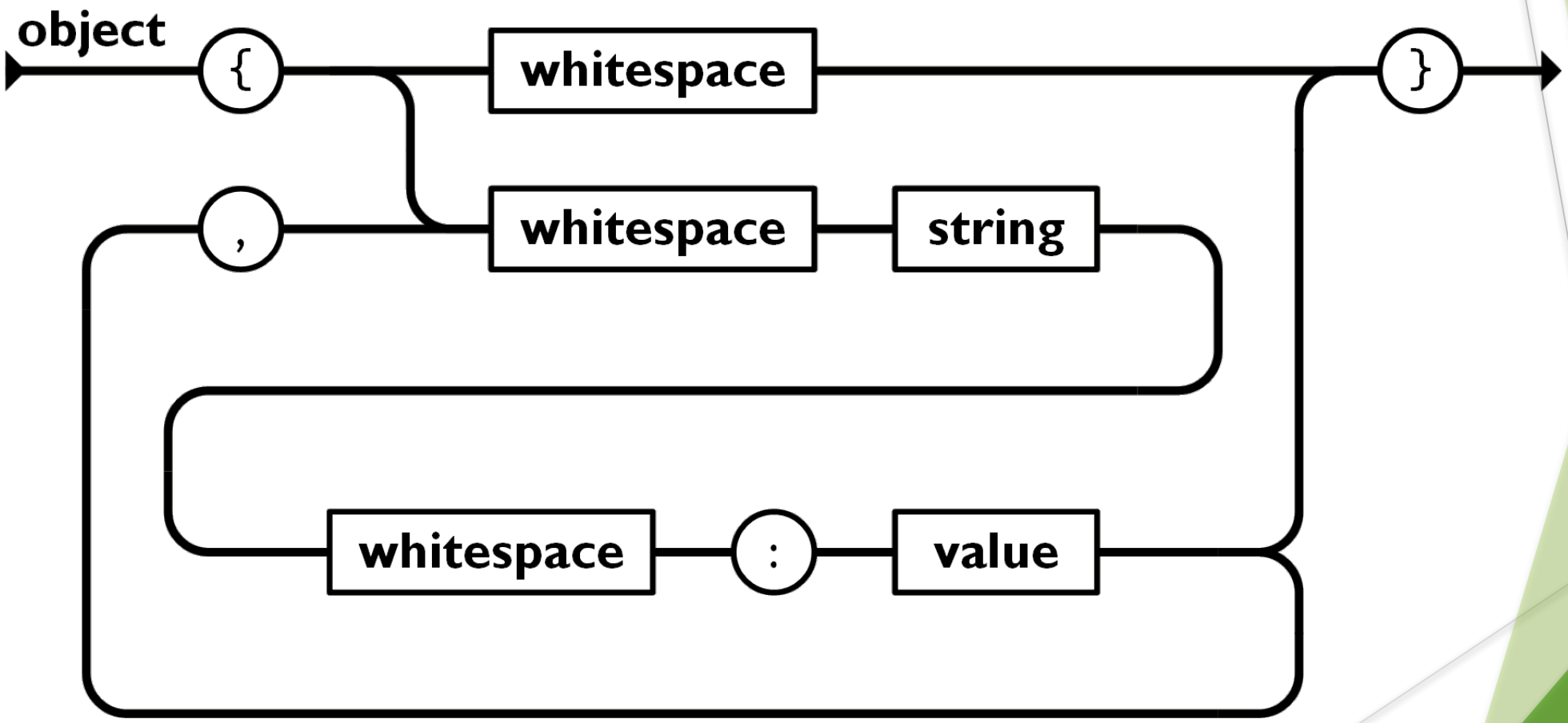
- a string
  - { "name":"John" }
- a number
  - { "age":30 }
- an object literal
  - {"employee":{ "name":"John", "age":30, "city":"New York" }}
- an array
  - {"employees":["John", "Anna", "Peter"]}
- a boolean
  - { "sale":true }
- null
  - { "middlename":null }

JSON data types



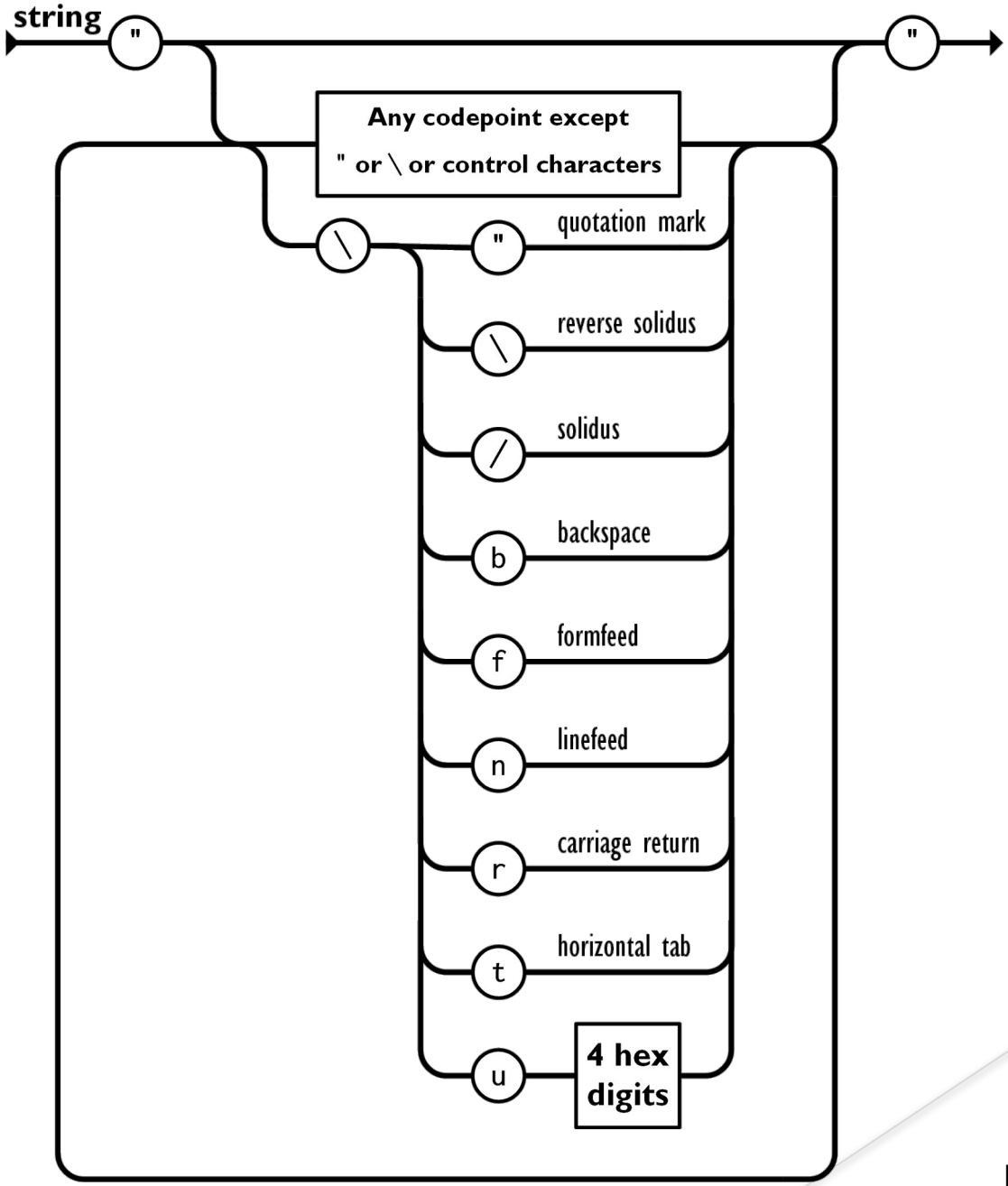
https://www.json.org/json-en.html

JSON object literal



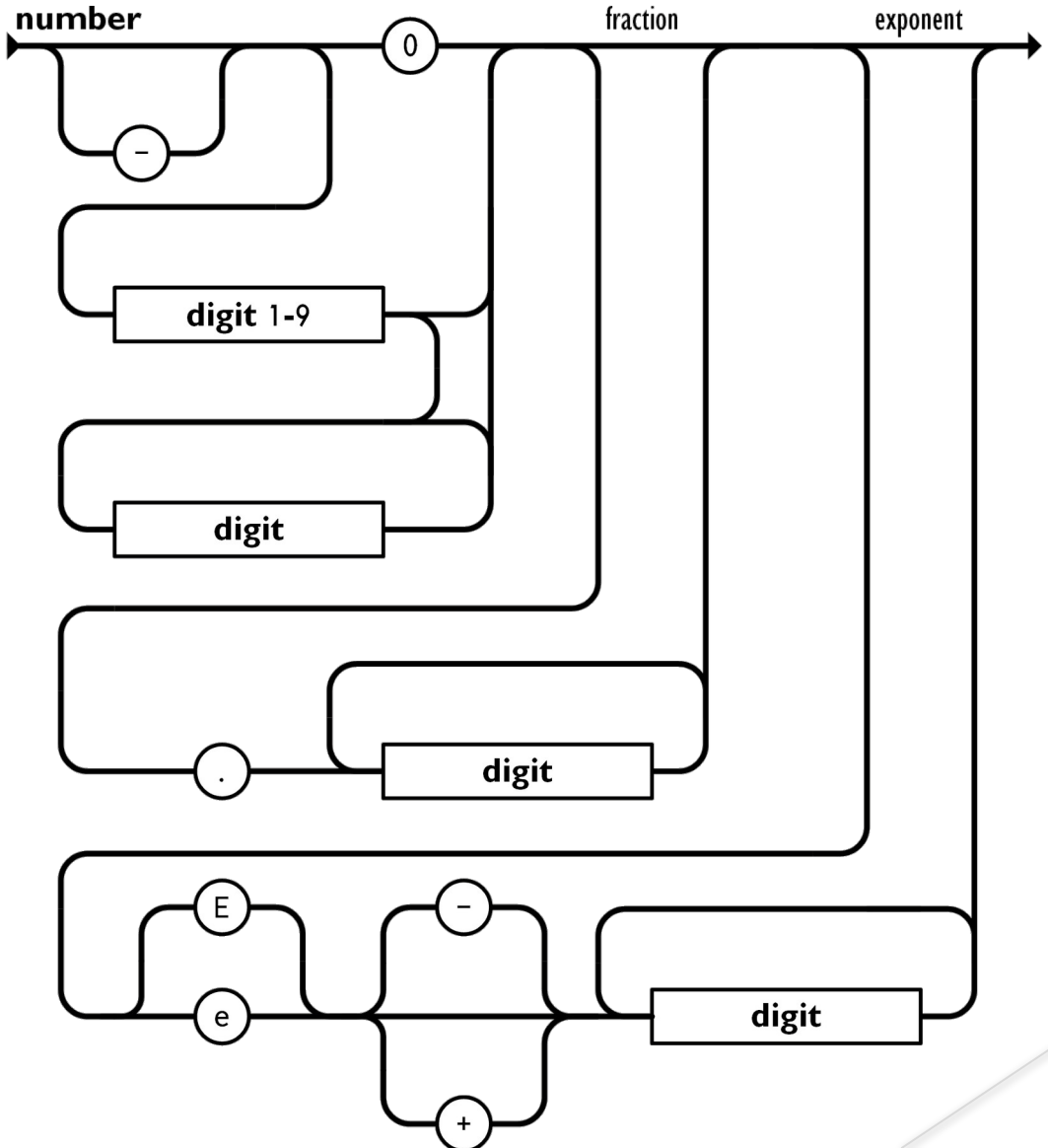
https://www.json.org/json-en.html

JSON String



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

JSON Number



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

How JSON works?

- The JSON format is syntactically similar to the code for creating JavaScript objects

- JavaScript native functionality can be used to convert to and from JSON format

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

- JS → JSON = JSON.stringify(JS)
- JSON → JS = JSON.parse(JSON)

JSON Parse

- A JavaScript string containing JSON syntax:

```
let text = '{"person": {"firstname": "john", "lastname": "smith"}}'
```

- The JavaScript function JSON.parse(text) can be used to convert a JSON text into a JavaScript object:

```
let person = JSON.parse(text);
```

Output:

JSON Stringify

- A JavaScript object:

  let person = {person: {firstname: "john", lastname: "smith"}}

- JSON.stringify() will convert objects to JSON text format

  let json = JSON.stringify(person);

Output:

```
json
'{"person":{"firstname":"john","lastname":"smith"}}'
```

Serialization

- Converting to and from JSON is called **Serialization**
- Serialization is the process of converting a non-contiguous collection of data into a contiguous form
- Non-contiguous data:
  - Objects
  - Data structures
- Contiguous data:
  - Byte stream

Serialization use

- Serialization is used to consolidate data for storing or transmission.
- Similar concept to archive files (.zip, .iso)
- Improves interoperability by standardizing data formats between technologies
  - EG: Website JavaScript can send data to a python API via a HTTP POST.

JSON and XMLHttpRequest

- A common use of JSON is to read data from a web server, and display the data in a web page.

- easy steps, how to read JSON data, using XHR
  - Create an array of objects in a web server
  - XHR GET request to server from the client
  - Receive data and set website element contents to the data

- The example reads JSON data from a web server running PHP and MySQL, and displays it in a <p> element innerText

Third Party APIs

- First Party APIs are integrated API services provided by the application or technology developer.
  - Web APIs like WebStorage, Geolocation
- Third Party APIs are APIs that provide non-standard services by other developers.
  - Services like Google Maps, Facebook SSO

Third Party API Interfacing

- APIs typically provide interface access via a common communication technology
  - HTTP GET, POST requests
- Performing a GET or POST request can be achieved in many ways
  - AJAX (XMLHttpRequest)
  - jQuery
  - Fetch
  - Axios
  - Many other third party options

Asynchronous code

- Code that requires waiting for a response to a request can (and should) be performed asynchronously.
- API interaction should be performed asynchronously to avoid blocking while waiting for a response.
- XHR, AJAX, Fetch support async with callbacks and promises.

AJAX XMLHttpRequest

```
const xhr = new XMLHttpRequest();
const url = 'https://api.endpoint.url/path';
xhr.open("GET", url);
xhr.send();

xhr.onreadystatechange = function () {
    if (this.readyState === 4) {
        console.log(this.resonseText);
    }
}
```

- Providing an event callback function lets the other code run while waiting for the request response. Once a response is received, the ready state changes, triggering the event.

jQuery AJAX

```
$.ajax({
    url: 'https://api.endpoint.url/path',
    dataType: 'json',
    success: function (data) {
        console.log(data);
    },
    error: function (e) {
        console.error(e);
    }
});
```

- Two callback functions are passed to the AJAX function, providing success and failure response code paths.

jQuery other methods

```
$.get(url, function(data, status) {
    console.log(data, status);
});

$.post(url, data, function(data, status) {
    console.log(data, status);
});

$getJSON(url, function(response) {
    console.log(response);
});
```

- These can all be achieved with $.ajax by specifying the dataType, or explicitly like above.

Promise vs Callback

- Promises are an alternative to callbacks.
- Instead of passing callback functions as parameters, we return a Promise object and attach responses to the Promise.

Callback passed to function

```
function processData(data, callback) {
    if (data) {callback("good")}
    else {callback("bad")}
}

processData("this is my data", function (response) {
    console.log(response);
})
```

Promise returned

```
function processData(data) {
    return new Promise(function(resolve, reject) {
        if (data) {resolve("good")}
        else {reject("bad")}
    })
}

processData("this is my data")
    .then(function(response) {console.log(response)})
```

Cleaner promise handling

```
processData("this is my data")
    .then(response => console.log(response))
```

Promises & Fetch

- With the introduction of "**Promises**" a new native XHR replacement is available: **Fetch**
- Fetch returns a promise, which is an asynchronous callback that can be handled using the "**then**" keyword:

```
fetch(url)
    .then(data => {return data})
    .then(response => {console.log(response)})
    .catch(error => {console.log(error)})
```

Fetch options

- Fetch can accept an options object to customise connection parameters:

```
fetch(url, {
    method: "GET", // POST, PUT, DELETE, etc.
    headers: {
        // the content type header value is usually auto-set
        // depending on the request body
        "Content-Type": "text/plain;charset=UTF-8"
    },
    body: undefined, // string, FormData, Blob, BufferSource, or URLSearchParams
    referrer: "about:client", // or "" to send no Referer header,
    referrerPolicy: "strict-origin-when-cross-origin", // no-referrer-when-downgrade...
    mode: "cors", // same-origin, no-cors
    credentials: "same-origin", // omit, include
    cache: "default", // no-store, reload, no-cache, force-cache, or only-if-cached
    redirect: "follow", // manual, error
    integrity: "", // a hash, like "sha256-abcdef1234567890"
    keepalive: false, // true
    signal: undefined, // AbortController to abort request
    window: window // null
});
```

Async and User Interfaces

- When waiting for async code to complete, the user interface should indicate an ongoing process to the user.
- A common way to do this is with a loading animation.
  - Scene loading (video games, slow websites): Throbber
  - Website elements: Spinner
  - Android apps: Indeterminate Progress Bar
  - iOS apps: Activity Indicator

Unreal Engine 4 throbber: https://cdn-ak.f.st-hatena.com/images/fotolife/s/shuntaendo/20210601/20210601230

Variety of CSS loaders: https://cssloaders.github.io/

Examples

https://jschollitt.github.io/week12/week12.html

Exercise

Moodle Week 12: Exercise – JSON, Third Party APIs