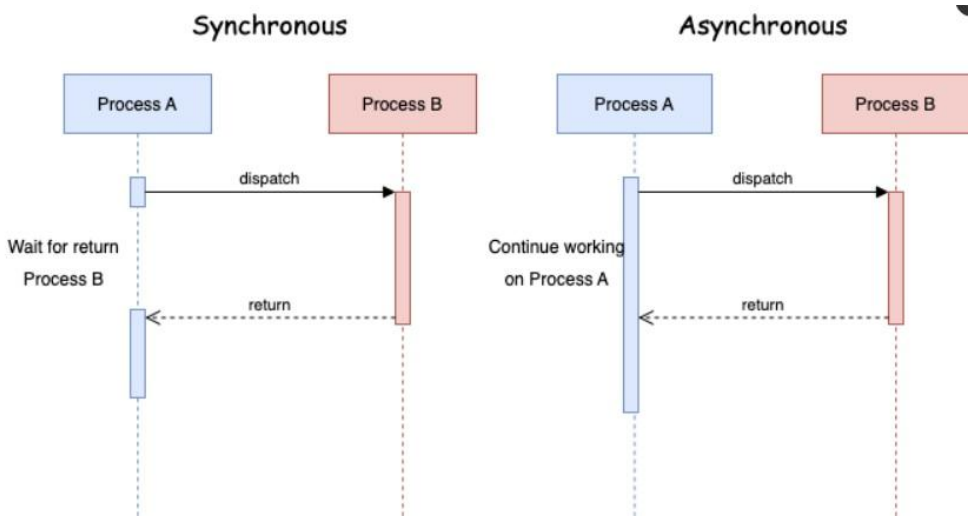
Communicating on the Web - Oct 26

Synchronous & Asynchronous JavaScript

Synchronous execution in JavaScript refers to events that occur sequentially, one after another. Synchronous loading of JavaScript files can be time consuming.

Accelerate load times with asynchronous loading.



Create an Asynchronous function

```
function fun1() {
      setTimeout(function () {
            msg.innerHTML += "<p>fun1 started :)</p>";
            msg.innerHTML += "<p>fun1 ended :)</p>"; },100); }

      function fun2() {
            msg.innerHTML += "<p>fun2 started :)</p>";
            fun1();
            msg.innerHTML += "<p>fun2 ended :)</p>"; }
      fun2();
```

Can also use setInterval BUT

Recall how to stop **setInterval**

```
Unset
var refreshIntervalId = setInterval(fname, 10000);

/* later */
clearInterval(refreshIntervalId);
```

Modified example from class

```
function fun1() {
           var myIntervalID = setInterval(function () {
              msg.innerHTML += "<p>fun1 started :)</p>";
              msg.innerHTML += "<p>fun1 ended :)</p>";
           },6000);

            setTimeout(function () {
              clearInterval(myIntervalID);
           },30000);
       }
      function fun2() {
           msg.innerHTML += "<p>fun2 started :)</p>";
           fun1();
           msg.innerHTML += "<p>fun2 ended :)</p>";
       }

       fun2();
```

Build a clock using the function setInterval()

```
function clock() {
          //code here
          const mydate = new Date();
          text.innerHTML = mydate.toLocaleTimeString();
      }
 clock();
```

Async & Await & Promise

We can implement asynchronous functions using the `async`, `await`, and `Promise` keywords.

By using the **async** keyword, we can make a function return a **`Promise`**, which can be resolved or rejected at a later time.

The **`await`** keyword is used to wait for the completion of a **`Promise`**. “await” is only valid inside the “async” function

Why use `await` if `async` is faster?

`async` allows for faster code execution, but it can cause issues when order or dependencies matter. `await` can pause execution until a process completes to ensure correct code execution and avoid errors.

PROMISES

Benefits:
→ Handling of asynchronous operations
→ Code readability
→ Error handling
→ Flow of control definition in asynchronous logic

Possible states of a promise:

fulfilled: the promise succeeded
rejected: the promise failed
pending: Promise is neither fulfilled nor rejected
settled: Promise is either fulfilled or rejected

Example:

```
var msg = document.getElementById("text");
var promise = new Promise(function (resolve, reject) {
      const continuePlaying = "Y";
      if (continuePlaying === 'N') { resolve('Your friend is happy!'); }
      else { reject('Your friend is boring!'); } });

promise.
      then(function (successMessage) {
            msg.innerHTML += "<p>" + successMessage + "</p>";
            console.log(successMessage); }).
      catch(function (errorMessage) {
            msg.innerHTML += "<p>" + errorMessage + "</p>";
            console.log(errorMessage); });
```

Then() & Catch()

Then() is performed when the promise is: resolve or reject.
Catch() is performed when the promise ends as reject or experiences an execution error

API

Application Programming Interface allows you to execute code from a different server than where your frontend or backend is.

API Classifications of SCOPE

- Private API
- Public API
- Partner API
- Composite API

Types of API

- SOAP (xml)
- REST (plain file)
- RPC (client executes a procedure on a server, server responds with output)
- WEBSOCKET (two way communication, JSON)

REST API

Representational State Transfer
Functions GET, PUT, DELETE and POST (HTTP verbs).
Stateless
Implementation of the client and server are independent

Client Server Communication

Making requests - A request generally consists of:

- an HTTP verb: GET, PUT, DELETE and POST
- a header (client passes information about the request)
- a path to a resource
- an optional message body containing data

Headers & Accept Parameters

Client sends a request in the header
Server sends a **type of content** (data) that the client can receive (understand/process).

Types of content:

- MIME Types (Multipurpose Internet Mail Extensions)
  MDN Web Docs.

MIME Types & subtypes

- image — image/png, image/jpeg, image/gif
- audio — audio/wav, audio/mpeg
- video — video/mp4, video/ogg
- application — application/json, application/pdf, application/xml, application/octet-stream
- Text — text/html, text/plain, text/css

Paths

Build the path for getting the **fellows** resource with id **43** that participates in the **lesson** with id **6** in the domain **correlationone.com**: `GET correlationone.com/fellows/43/lesson/6`

Responses

The server response sends a data payload to the client. This must include a content-type in the header with the goal of indicating what type of data is being sent. (MIME TYPES)

Response example:

```
HTTP/1.1 200 (OK) Content-Type: text/html
```

Status code	Meaning
200 (OK)	This is the standard response for successful HTTP requests.
201 (CREATED)	This is the standard response for an HTTP request that resulted in an item being successfully created.
204 (NO CONTENT)	This is the standard response for successful HTTP requests, where nothing is being returned in the response body.
400 (BAD REQUEST)	The request cannot be processed because of bad request syntax, excessive size, or another client error.
403 (FORBIDDEN)	The client does not have permission to access this resource.
404 (NOT FOUND)	The resource could not be found at this time. It is possible it was deleted, or does not exist yet.
500 (INTERNAL SERVER ERROR)	The generic answer for an unexpected failure if there is no more specific information available.

HTTP Verb expected status code

GET => return 200 (OK)
POST => return 201 (CREATED)
PUT => return 200 (OK)
DELETE => return 204 (NO CONTENT)

Exercise

Build the path for getting the **fellows** resource with id **43** that participates in the **lesson** with id **6** in the domain **correlationone.com**:

```
GET correlationone.com/fellows/43/lesson/6
```

Exercise

Build a response to the following request:

```
POST https://correlationone.com/fellows Body: { "fellow": { "name" = "Harry Musah", "email" = "harry.musah@gmail.com" } }
```

Response

HTTP/1.1 201 Created
Content-Type: application/json
Location: https://correlationone.com/fellows/123


```
{
  "fellow":{
    "id": 123,
    "name": "Harry Musah",
    "email": "harry.musah@gmail.com",
    "created_at": "2023-10-26T15:30:00Z"
    }
}
```