1. What does the fetch() method do? How can it be used instead of an AJAX call?

Fetch is an API for web browsers that allows for the asynchronous loading of text, images, and structured data in order to update an HTML page. This is similar to the concept of Ajax, but fetch utilizes the Promise object, which makes the code simpler, particularly when used in conjunction with async/await. While one could use promises with the XMLHttpRequest object, this would make the code even more complex, resulting in a larger program. Fetch works with most recent browsers, except for Internet Explorer. In situations where maximum compatibility is necessary, Ajax should continue to be used to update web pages. If interaction with the server is also needed, the WebSocket object is more appropriate than fetch. However, in other cases, fetch provides a simpler way to load content onto a page, as will be demonstrated below.

a. Loading a text:
   To load a text using Fetch, a resource is retrieved and then converted into a string using the text() method. Since both of these actions return a promise, the result must be awaited before proceeding. For example,

```
<fieldset>
<p id="textdemo"></p>
</fieldset>

<script>
var textdemo = document.getElementById('textdemo');

async function loadText(fname) {
        var str = await fetch(fname)
        textdemo.innerHTML = await str.text()
}

loadText("demotext.txt")
</script>
```

b. Loading an image:
   When using Fetch to load an image, a blob is returned through the blob() method. However, in order to assign the loaded image to the src attribute of an img tag, a URL is required. To obtain a URL from the content retrieved by fetch, we must use the createObjectURL method of the URL object. For example,

```
<fieldset>
<img id="imgdemo" />
</fieldset>

<script>
var imgdemo = document.getElementById('imgdemo');

async function loadImage(fname) {
        var response = await fetch(fname)
        imgdemo.src =  URL.createObjectURL( await response.blob() )
}
```

```
loadImage("images/angry-birds.jpg")
</script>
```

c. Loading a JASON file:

Fetch can be used to load a JSON file as a resource, and then the json() method can be used to parse it into a JavaScript object. Once this is done, the values of the object's attributes can be accessed using their respective names. For example,

```
<fieldset>
<p>Name <input type="text" id="jsondemo1" value="" /></p>
<p>Year <input type="text" id="jsondemo2" value="" /></p>
</fieldset>

<script>
async function loadJSON(fname) {
        var response = await fetch(fname)
        var j =  await response.json()
        document.getElementById('jsondemo1').value = j.name
        document.getElementById('jsondemo2').value = j.year
}

loadJSON("demojson.json")
</script>
```

d. Adding Controls:

It is possible to handle the situation where the resource being loaded is not accessible on the server. While there are various methods to control the return of Fetch, the easiest way is to use a try-catch statement. For example,

```
<fieldset>
<p id="ctrldemo"></p>
</fieldset>

<script>
var ctrldemo = document.getElementById('ctrldemo')

async function loadCTRL(fname) {
        try {
                var response = await fetch(fname)
                var j =  await response.json()
                ctrldemo.innerHTML = JSON.stringify(j)
        }
        catch(err) {
                ctrldemo.innerHTML = "Error, can't load " + fname + " " + err
        }
}

loadCTRL("demojson.json")
</script>
```

Reference: https://www.xul.fr/en/html5/fetch.php

2. In JQuery, what does it mean to "traverse the DOM"?

DOM traversal refers to the process of moving through the elements of a web page relative to a selected element or set of elements. While traversing, it is possible to replace the initial selection with a new one or add and remove elements from it.

To make a selection more specific, filtering elements can be applied. Elements can be filtered based on various conditions such as their position in relation to other elements or whether or not they possess a specific class. Typically, this process results in a smaller number of selected elements than what was originally present.

The following is a list of different methods that can be used for filtering elements:

- **eq** method: reduces the selected set of elements to one located at a specific index. The index is zero-based, so to select the first element, the syntax would be $("selector").eq(0). Starting from version 1.4, a negative integer can be used to count elements from the end instead of the beginning.
- **first** and **last** methods: return the first and last elements, respectively, from the set of matched elements. These methods do not accept any arguments.
- **Slice()**: it can select all elements within a given range of indices. It takes two arguments: the starting index and the ending index of the selection. The ending index is optional, and if not provided, all elements whose index is greater than or equal to the starting index will be selected.
- **Filter**: it is used to narrow down the set of elements to those that match the specified selector or criteria set by a function passed to the method. Here is an example of using this method with selectors:

```
$("li").filter(":even").css( "font-weight", "bold" );
```

- The **map** method: allows you to apply a function to each element in the current selection, creating a new jQuery object that contains the return values. The resulting jQuery object contains an array that can be accessed using the get method to work with a basic array.

In a situation where you have access to a selection of elements using a specific selector, but you need to work with their parents, the parents do not have a common tag or class, but they all have a commonality in that they are the parents of the elements you have access to. This type of situation is quite common and can be addressed using various methods provided by jQuery to access parents, children, or siblings.

These methods include **children, find, parent, parents, and closest**. Children method retrieves the direct children of the selected elements, while find method retrieves all descendants filtered by a selector or element. Parent method gets the direct parent of the selected elements, while parents method gets all the ancestors. Finally, closest method gets the first element that matches a given selector by testing the element itself and traversing up in the DOM tree.

Reference: https://www.sitepoint.com/comprehensive-jquery-dom-traversal/