JavaScript
Intro for Beginners
Lab 5

1 Working with JSON

File to use:
`json-basic.js`
`sample student marks.xlsx`
`superhero.json`

JavaScript Object Notation (JSON) is a widely used format of exchanging data between applications, particularly over the Web. Although JSON is based on JavaScript objects, it is designed to be used independently of JavaScript (or any other programming language). All major programming languages have libraries or functions that specifically deal with processing JSON data.

It is always a good idea to validate the JSON that you are working with to prevent unexpected errors when executing code to parse or process it.

There are several main ways of working with JSON in JavaScript directly. We can also read from a JSON file and convert it into an object and vice versa.

You can save an existing Excel spreadsheet into CSV form, and then convert it into JSON form.
You can then read the JSON data and convert it into a JSON object.

As an exercise (using `sample student marks.xlsx`), you could try to replicate the functionality of common Excel formulas using a JavaScript program, for e.g.:

- Find the total / average mark for each student
- Find the average mark for each subject
- Find the 5 students with the highest / lowest marks for each subject
- Sort the students based on the mark for a given subject or average mark for all subjects

2 Try - Catch - Finally

File to use:
`try-catch-basic.js`
`throw-basic.js`

The try-catch block is used to handle run time exceptions that may occur, so that the program can still continue running instead of being terminated immediately. The try-catch-finally block provides a block that will be executed regardless of whether exceptions occurred or not. We can always use the throw keyword to create our own custom Exception, to signal the detection of an issue that would count as an error in our code logic.

3 Promises

Files to use:
`problem-async-demo.js`
`promises-anonymous.js`
`promises-arrow.js`

Promises are is the main way we deal with asynchronous operations in JavaScript. A Promise is an object that is created by calling the Promise function and passing it a single function (called the executor) as its parameter. This function itself takes two parameters which are also functions: resolve and reject. When the `then` method is called on the Promise object, this method takes another two more callback functions as its parameters. These callbacks are passed to the resolve and reject parameters.

4 Using the Fetch API

File to use: `fetch-basic.mjs`

The Fetch API is already part of the JavaScript engine in modern browsers, so you can use it directly within a script referenced from a HTML document in a browser. For the case of a JavaScript program executed directly from Node.js, we first need to install the node-fetch package using NPM.

NPM is a package manager which is used for managing dependencies for a Node.js project. The dependencies are what the specific project needs to run correctly and usually consists of packages: which are a collection of JavaScript programs that perform functionality related to that project.

In a command prompt in the folder containing this file, type:

```
npm install node-fetch
```

The files for this package is installed in a newly-created subfolder `node_modules` within the current folder, and two configuration files `package.json` and `package-lock.json` are created as well. These files provide description about the project that is to be run in the current folder and any metadata related to it, such as the dependencies it needs to use.

Notice that the extension of this source code file is `mjs` instead of `js`, to indicate that it is a JavaScript module based on the ECMAScript Module system. This is necessary for the Fetch library API to run correctly.

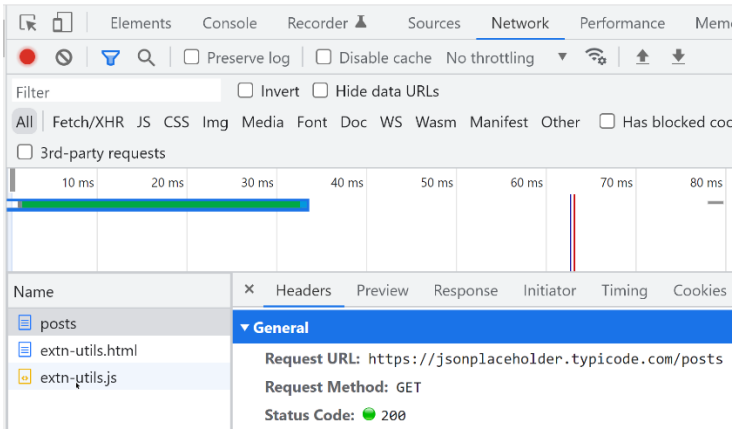
The Fetch API is a modern library that allows a JavaScript program to make HTTP requests to a web server. Typically the request will be made to a REST API using standard HTTP methods such as GET, POST, PUT, PATCH and DELETE. The most commonly used methods will be GET and POST.
The Fetch API is based on promises.

We can use a sample fake API service to return JSON responses to test our JavaScript programs. You can check the guide on the various kind of resources you can retrieve using the Fetch API.

You can initially test out sending HTTP Get requests to these URLs by simply typing them into the address bar of the browsers, for e.g.

https://jsonplaceholder.typicode.com/posts
https://jsonplaceholder.typicode.com/posts/1
https://jsonplaceholder.typicode.com/posts/5

If you view the Network panel of the Chrome Developer Tools, you will be able to see the GET request in the Headers sent out as well the JSON sent back in the response message:



You can then test out the Fetch API by running the related JavaScript file in the usual manner:

```
node fetch-basic.mjs
```