JavaScript JSON AJAX and Objects

JavaScript Objects

JavaScript Objects Example of how to use Objects in Code get object data

JSON is based upon JavaScript syntax and easily converts into a JavaScript object. This is why learning how to use objects and arrays in JavaScript can help better understand JSON. JSON is a syntax that is used to serialize objects, arrays, numbers, strings, boolean and null values.

Some important key differences between JavaScript objects and JSON

- property names must be double quoted
- trailing commas are not allowed
- JavaScript functions are not allowed
- Leading zeros should not be used, and decimal points must be at least one digit after them
- JSON is a string with a specified data format and it contains only properties, no methods.

Any JSON is valid JavaScript but any JavaScript object is not valid JSON. You can use the JSON methods in JavaScript code to parse JSON converting from string to object, and also object to string. JSON parse will convert a string with JSON syntax into a usable JavaScript object. JSON stringify will convert a JavaScript object into a String.

JSON is an object which can be used to describe something and hold related data.

```
{
 "car1": "black",
 "car2": "blue"
}
```

Data Types in JSON can be any of these :

Number - can be digits, positive or negative,decimal {"name":10}

String - double-quoted {"name":"Hello world"}
Boolean - true or false {"name":true}
Array - Sequence of values separated by commas, uses square brackets. Indexing starts with 0. {"name": [{"name1": 1},"hello", "world"]}
Object - unordered collection with key:value pairs. Wrapped with curly brackets {}. Colons are used to separate the key and name. Keys should be strings and have unique names. {"name": {"name1": 1,"name2": 1}}
Null - empty value {"name": null}

Basics of JSON

- JSON is extended from the JavaScript objects
- Uses the media type application/json
- File extension is .json

key/name value pairs { "name" : "value" }
Items are comma separated { "name1" : "value" , "name2" : "value", "name3" : "value"}
Arrays use square brackets with values separated by commas { "name" : [{ "name" : "value" }, { "name" : "value" }] }

Objects in JavaScript can contain multiple values in a key name value pair structure. Name value pairs must be separated by a comma. The name and the value are separated by a colon. The property names in that object are unique and cannot be used more than once, or the last assigned value to the property name will be used as the value for the name.

Objects can be used to hold data, the object is referenced in the code by the variable name. The variable name is the main container, which then can be used to return the values contained within.

There are two ways to get values from an object, dot notation where the object variable and the property name are user, separated by a dot. The second way is

using the bracket notation, which uses the object name and a set of brackets wrapping the property name. Both will return the same value, although to use property names with a space you must use the bracket notation. Dot notation (obj1.first) vs bracket notation (obj1[‘first’])

Dot notation vs Bracket Notation

```
const myJSON = {}
myJSON.car1 = "black"
myJSON["car1"] = "blue"
console.log(myJSON)
```

An object is used to hold multiple values, as a collection of related data. Functions can be contained within objects, which are then referred to as methods. These are only available within JavaScript objects and not within JSON as the JSON is static data. Create an object by creating a variable name and assigning the object using the {} to the variable. Object names can hold values of other objects and arrays, which make them ideal for holding large amounts of data that is human readable. Can go multiple levels deep, as many as needed.

Exercise : Create and object

1. Create an HTML file, link the JavaScript file to the HTML.
2. Open the HTML in your browser.
3. Within the browser open the devtools to view the JavaScript output
4. Create a variable and assign the object to it.
5. Within the object, add several values with different data types.
6. Output the object into the browser console using the console.log() method.
7. Select the values contained within the object, try using the bracket notation as well as the dot notation.

```
<!doctype html>
<html>
<head>
```

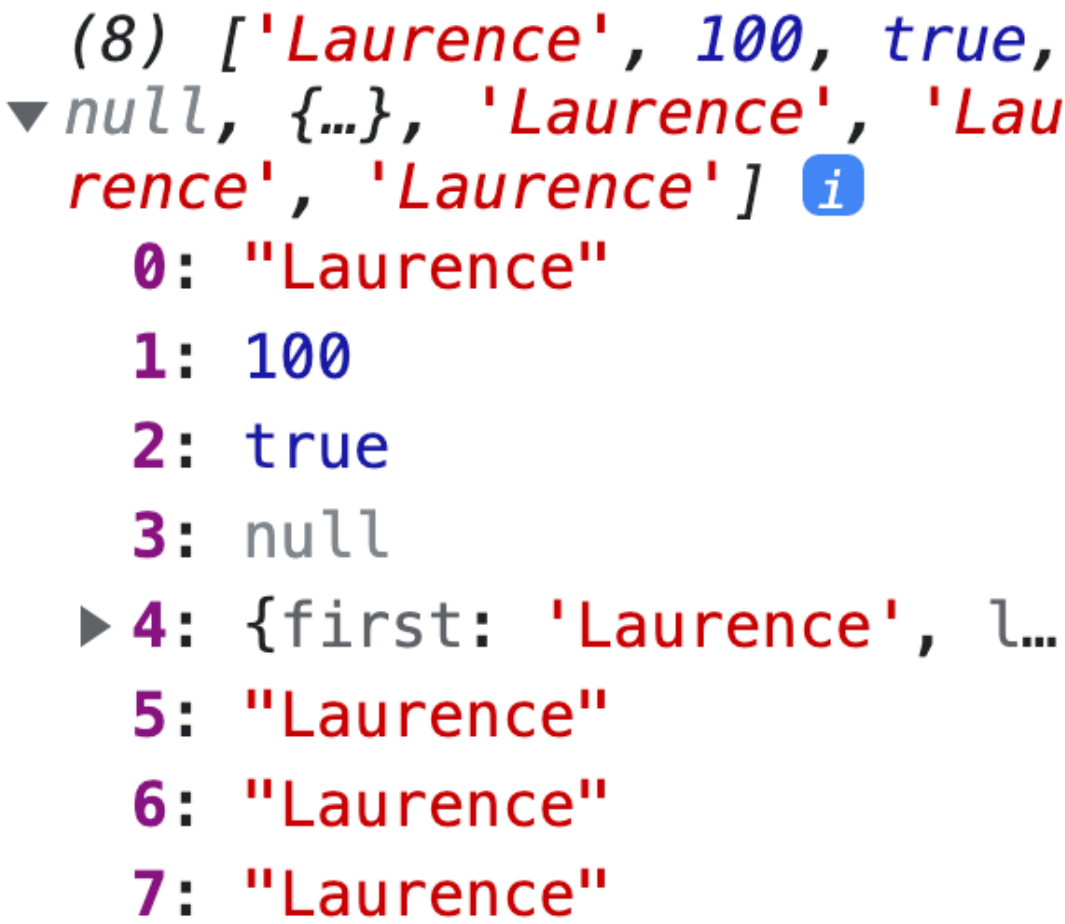
```
  <title>JavaScript JSON AJAX</title>
</head>
<body>
  <div></div>
  <script src="apps1.js"></script>
</body>
</html>

const obj1 = {
   "first" : "Laurence",
   "last name" : "Svekis",
   "num" : 100,
   "boo" : false,
   "nu" : null,
   "arr" : [],
   "obj" : {}
};
console.log(obj1);
console.log(obj1.first);
console.log(obj1["first"]);
console.log(obj1["last name"]);
```

JavaScript Arrays

JavaScript Arrays examples of using arrays within code items and data in array

Arrays provide a way to hold multiple values in one container. In JavaScript arrays come with powerful methods that can be used within the code to interact with the contents of an array.

Items in the array are comma separated, index values increment by one and do not get skipped.



Example of an Array and its contents within the browser console.

The items contained within the array can be of different data types, including strings, numbers, booleans, arrays and objects. The values of the items in the array can be retrieved using the index value of the item, these will all be unique. Arrays are zero based, which means that the index values of an array start at 0 and increment up.

Exercise : Create an array and add data into the array.

1. Create an array
2. Add different values into the array, by comma separation.
3. Output the array into the console, open the console values to view the contents.
4. Select the values from the array and output them into the console.
5. Add an object into the array, retrieve the object values using the property names and output them into the console.

```
const arr = ["Laurence", 100, true, null, {
   "first": "Laurence",
   "last": "Svekis"
}, "Laurence", "Laurence", "Laurence"];

console.log(arr);
console.log(arr[5]);
console.log(arr[4]["first"]);
console.log(arr[4].last);
```

Updating Objects and Arrays

How to use objects and arrays in JavaScript code to output data.

Arrays have a data type of object within JavaScript. Although arrays and objects have different properties in the code, and different ways to interact with the contents of these data objects.

They both share the way that they are held within the browser, the data is stored in a memory location which is different from the value if assigned to the variable. This is why we can use const to declare the variable that is assigned to the object and still select the contents and make updates to the data within.

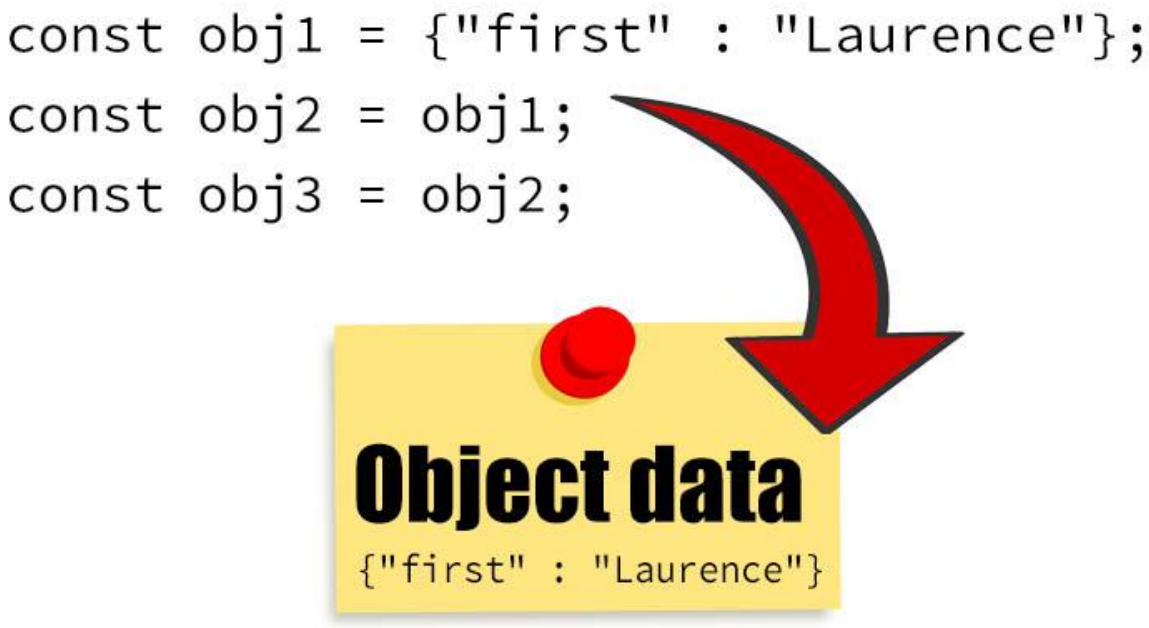


Illustration of how objects data is assigned to a variable.

You can nest both objects and arrays within other objects and arrays. You can even nest themselves within them, although this is something that should be avoided. This is only possible within the code and cannot be represented within JSON as the

JSON structure is static and the syntax needs to have all the values set. Variables in code can be used to represent the value, and when using objects those values can then be assigned to other variables or in the case of objects the location where the object data is stored can be assigned to the variable.

When assigning different variables to the same object, or assigning the object using the variable name, this will result in having multiple references to the same data object. This should be avoided as it can lead to confusion when reading the code.

Arrays have a property of length, which indicates the number of items within that array. If you assign a new value to an item with a larger index value than the last item or containing items, the array will automatically add empty items to the index values that were skipped. If you update the length property and set a new value to it, this will change the number of items in the array. If you update the length to be less than the current items, those will be removed. If you set the array length to be zero it will clear everything from the array!

Exercise : Fun with objects and arrays get the values.

1. Create an object with an array as one of the values.
2. Within the array nest a second array, to make it challenging to retrieve the items within the array.
3. Using the object, get the value of the item in the array and output it into the console.
4. Using the same syntax you used to retrieve a value, assign a new value to the item in the array.
5. Note the length property of the array, assign a value to an item with a larger index value than the array length.
6. Reassign a new value to the length of the array, see what happens to the items in the array.
7. Assign an existing array into a new index value of an array. Update the values of the items and check the results. Notice the connection and how it works between the arrays. They have the same values.

8. Add arrays into objects, and objects into arrays to practice retrieving the values.

```
const obj1 = {
   "first": "Laurence",
   "id": 1000,
   "courses": ["html", "css", "js", [1, 2, 3]],
   "first": "Mike",
   "first": "Laurence"
};
obj1.first = "John";
obj1.courses[0] = "json";
obj1.courses[10] = "html";
console.log(obj1.first);
console.log(obj1.courses[0]);
console.log(obj1.courses[1]);

console.log(obj1);
console.log(obj1.courses.length);
obj1.courses.length = 4;
console.log(obj1);
console.clear();

console.log(obj1.courses[3][0]);
const arr1 = ["one", "two", "three"];
console.log(arr1);

obj1.courses[4] = arr1;
console.log(obj1);
```

```
let a = 1;
a = 10;

console.log(obj1.courses[4][2]);
obj1.courses[4][2] = "five";
obj1.courses[4][4] = "six";
arr1[0] = "NEW";
console.log(obj1);
console.log(arr1);

const obj2 = {
   "first": "Hello",
   "arr": arr1
}
console.clear();
console.log(obj2);

arr1[9] = arr1;
```

Loop through the data

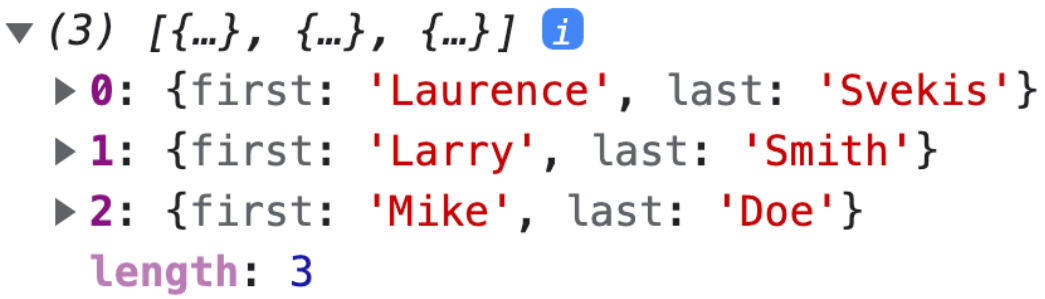
Get items in arrays and objects output the data using JavaScript for loops and iteration of objects.

Both objects and arrays are able to hold lots of content. They can be many levels deep with arrays and objects nested within each other. JavaScript has several ways to iterate through the content in order to get to the data. You can loop through the content in a number of ways using JavaScript as well as converting object keys and values into array structures which can then be iterated to get to the contents.

Arrays need the index to return the value associated with it. The index values are sequential which makes it easy to select the next item in the array. We can get the length of the array as its already a property for the array.

When creating arrays that contain objects that are intended to be grouped and output with iteration, keep data structured the same so that you can easily determine where the values are located. If all the objects that are items in the array share the same key values, then these can be coded and selected as we loop through the contents. No surprises we should know what the keys are in order to loop through the array items and output all of them consistently. This is important to plan when creating objects that will be used in this way.

The forEach() method executes a block of code within the function once for each array element.



Example of an array with items as objects, sharing the same structure of key name values. This makes it easier to iterate the object data as you can anticipate what the key will be.

Exercise : Get the data from an array and object

1. Create an object, output the results for the Object.entries(obj1) in the console.
2. Using the for..of loop through the contents of the object, return both the key names and the values into the console.
3. Using Object.keys() and Object.values() output the results into the console for the object.
4. Create an array, and create two more arrays with the results of the Object.keys() and Object.values()
5. Loop through the array using its length value, and a typical for loop.
6. Use the for…in loop to get all the index values of the items in the array, then using the index values output the item values into the console.
7. Using forEach() array method, loop through the contents of an array. Output the index, the item value, and the array itself into the console.
8. Add an array into the object, and loop through the array items as they are contained in the object.
9. Create a new array, with objects as the items. Structure them with the same key values, so that you can output the results. Add key values of first and last.
10. Using for each output just the last names into the console from the array. Then output just the first names into the console.

```
const obj1 = {
   "first": "Laurence",
   "last": "Svekis",
   "id": 100
}
console.log(Object.entries(obj1));
for (const [key, value] of Object.entries(obj1)) {
   console.log(key);
   console.log(obj1[key]);
   //obj1[key] = 'Smith';
}
const arr1 = (Object.keys(obj1));
const arr2 = (Object.values(obj1));
const arr3 = ["Laurence", "Svekis", 100, true, "hello world"];

console.clear();
console.log(arr1);
console.log(arr2);
console.log(arr3);

for (let i = 0; i < arr3.length; i++) {
   console.log(i, arr3[i]);
}

for (i in arr3) {
   console.log(i, arr3[i]);
}
```

```
arr3.forEach((item, index, array) => {
   console.log(item);
   console.log(index);
   console.log(array);
})

Object.values(obj1).forEach((val) => {
   console.log(val);
});

Object.entries(obj1).forEach((val) => {
   console.log(val[0], val[1]);
});

//console.clear();
obj1.arr = arr3;
console.log(obj1);

obj1.arr.forEach(el => {
   console.log(el)
});

const friends = [{
       "first": "Laurence",
       "last": "Svekis"
   },
   {
       "first": "Larry",
```

```
        "last": "Smith"
    },
    {
        "first": "Mike",
        "last": "Doe"
    }
];

//console.clear();
console.log(friends);

friends.forEach(friend => {
    console.log(friend.last);
})

friends.forEach(friend => {
    console.log(friend.first);
})
```

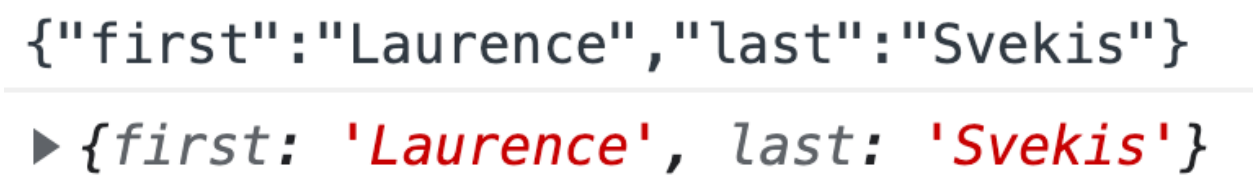
The JavaScript JSON Object

Explore JSON and how you can get JSON data as a JavaScript Object

JSON is a language independent data format which was derived from JavaScript objects. JSON is the most commonly used data format on the web, with many modern programming languages able to use it. JSON (JavaScript Object Notation)

JavaScript has an object called JSON which can be used to parse a string JSON structured value, into a JavaScript Object. There are two method properties within the JSON object, one to parse a string into an object, the other to convert an object into a JSON structured string.

```
const myStr = '{"first":"Laurence","last":"Svekis"}';
const myObj = JSON.parse(myStr);
console.log(myStr);
console.log(myObj);
```



Output from a JSON.parse of a string and the string in the console.

The JSON.stringify() method converts a JavaScript object or value to a JSON string like {"first":"Laurence","last":"Svekis"}
The JSON.parse() method parses a JSON string which needs to be properly structured as a JSON object and constructs the JavaScript value contained in the string.

Exercise : Convert an object into a string and back to an object

1. Create an array with several objects as the items

2. Using the JSON.stringify() convert the object into a string value. Note the string cannot be used to select the contents or data as the data type is now just a regular string.
3. Take the string that is JSON structured and convert it back into an object, JSON.parse(). Select some values from the object.
4. Add a new item into the array of the converted string, notice that it is no longer connected to the original object and is a totally separate instance.
5. Update the string wrapping it with [] and {}. Which one works, and why? The brackets work as it nests the array within a new array, the object brackets won't work as the object does not contain a property name. To use parse you must have a properly structured string which represents the object.
6. You can try the string within a validator application, validating the JSON will check for proper JSON structure as well most will space the data so it can be easier to read through it.
7. Try several other examples of converting objects to strings, and strings to objects.

```
const arr1 = [{
        "first": "Laurence"
    },
    {
        "first": "Mike"
    },
    {
        "first": "Jane"
    },
    {
        "first": "Lisa"
    },
```

```
    {
        "first": "Henry"
    }
];

console.log(arr1[0].first);
const str1 = JSON.stringify(arr1);
console.log(str1);
console.log(str1[0]);
const jsObj1 = JSON.parse(str1);
console.log(jsObj1[0].first);
arr1[5] = {
    "first": "Joe"
};
console.log(arr1);
console.log(jsObj1);
const str2 = '[' + str1 + ']';
console.log(str2);
const jsObj2 = JSON.parse(str2);
console.log(jsObj2[0][0].first);
const obj2 = {
    first: 'Laurence',
    last: 'Svekis'
}
console.log(obj2);
const str3 = JSON.stringify(obj2);
console.log(str3);
const jsObj3 = JSON.parse(str3);
```

```
console.log(jsObj3);
```

JSON files and Contents with JavaScript

Use JSON data to connect to JSON files and get contents with JavaScript code.

To use JSON data the contents of the json file need to be valid JSON. The file contents can be retrieved as regular text and then converted into an object or as a JavaScript Object directly from a valid JSON file. JSON files should use the .json extension for the file, and only contain JSON data.

```
{
   "friends": [
       {
           "first" : "Laurence",
           "last" : "Svekis",
           "id" : 100
       },
       {
           "first" : "Mike",
           "last" : "Jones",
           "id" : 44
       },
       {
           "first" : "Jane",
           "last" : "Doe",
           "id" : 15
       },
       {
           "first" : "Jack",
           "last" : "Jones",
```

```
            "id" : 15
        }
    ]
}
```

Connect to a file using the JavaScript fetch API. The fetch() API is asynchronous and provides an interface for fetching files across the network. The files can be local or file from other servers, you will need the path for the file either relative or absolute depending on the location of your JavaScript file. Fetch API provides a more powerful and flexible set of features of XMLHttpRequests. For making a request and fetching a resource, use the fetch() method.

In the fetch() method there is only one mandatory argument which is the path to the resource. It is promise based which means that it waits for the response back from the server. This is important since unlike when you have an object in your code which is ready to be used, the data coming from the end point might not come in right away, this would cause a lag in the code if we had to wait. Fetch solves this by having the promise built in when the response resolves.

Once fetch retrieves a response there are a number of methods available to define the response content. Response properties include the body, which is the body of the contents, the headers, status, type, url and others. The response object also has methods like, text() which resolves a text representation of the response body, and json() which resolves the content parsing the response body as JSON.

For the fetch API you can also include options to set the method which by default is GET and setting headings as well as additional options.

Exercise : Connect to a JSON file and output the contents into the console.

1. Create a JavaScript object with an array of items that are the same structured objects. You can use the example in the above JSON.

2. Create a function that will loop through all the items in the object array, and output the item object property values.
3. Create a file with the extension of json, and copy the entire JavaScript object you just used into the file.
4. Create a new function to run the fetch connection, add the file path in the () rounded brackets as an argument in fetch.
5. Add the first promise using then() and return the content back as text()
6. In the second then() get the data and using JSON.parse() convert the text string into a JavaScript object.
7. Send the new JavaScript object into the same function that used the original object outputting the results of the data from the JSON file into the console.
8. Create a second function that will make a fetch request to the same JSON file, this time use the response method of json() and return the JSON data as a usable JavaScript object. Send the data to be output like previously. The output results should be the same as the previous 2 examples.

```
getData();
function getData(){
   const url = 'json6.json';
   fetch(url)
   .then((rep)=>rep.json())
   .then((data)=>{
       console.log(data);
       output(data);
   })
}

function getData2(){
   const url = 'json6.json';
   fetch(url)
```

```
    .then((rep)=>rep.text())
    .then((data)=>{
        console.log(data);
        const jsObj = JSON.parse(data);
        console.log(jsObj);
        output(jsObj);
    })
}

function output(json){
    json.friends.forEach((person,index)=>{
        const temp = `${person.first} ${person.last} ${person.id}
`;
        console.log(temp);
        //const str1 = `"${4+5+index}     "'Test'`;
        //console.log(str1);
    })
}
```

Async and Await and Promises

JavaScript asynchronous requests using async and await promises

The fetch() method is a process of requesting a resource from the network, and returning back the promise. Another way to create promises is using async and await. Promises will run after the initial code is stated, which allows time for resolving the response to the request. Using promises is ideal for situations where data is not available right away, and you want to asynchronously make the request and then handle it once the data is returned. In fetch the promise resolves to the response object, which can then be used to return the content.

The keywords Async and await can be used for asynchronous functions. Placing the async keyword in front of a function can turn it into an async function. The function will now return a promise back. Adding the await keyword can be used to pause the code until the promise fulfills. If we use it with fetch, once the fetch request fulfills we can continue with the code.

Using catch() method in the promise allows you to handle any errors that might occur. You can also use the try, catch to throw custom errors but this is typically not needed when using catch.

Within fetch if you use the chaining of the promises, once the promise resolves we move to the next step, which can then run a function to output the results of the response object. This provides an easy to read simple syntax that can be used to wait for responses, catch errors and handle response data, especially JSON data since fetch has the method to convert the JSON structured response object into a usable JavaScript object.

```
fetch(url)
    .then(rep => rep.json())
    .then(data => viewData(data))
```

```
        .catch(error => console.log(error));
```

Exercise : Explore how promises work.

1. Create a JSON file and get the URL as a string value
2. create a function to make a fetch request, in the function called output2() output into the console the word ‘one’. Then make the fetch request and output the response data into a function called viewData
3. Create a function called viewData which outputs a val from the argument into the console.
4. Create a function called output(), invoke the function. Within the function add the keyword async and use await to pause the code until a response is retrieved from the fetch request. Create a new request object with the url and add that into the fetch request. In the console output the word ‘five’
5. Create a third function named output1 which returns a value of ‘three’, using console.log() invoke the function so the return results are output into the console.
6. Add a line that outputs the string value of ‘four’
7. Run the code, notice the order that the console outputs, the promises will output after the other code.
8. Catch and throw errors using the catch() method and the try and catch keywords.

```
const url = 'json6.json';
output2();
output();
console.log(output1());
console.log('four');
function output1() {
   return 'three';
}
```

```
function output2() {
   console.log('one');
   fetch(url)
       .then(rep => rep.json())
       .then(data => {
           viewData(data);
       })
       .catch(error => console.log(error));
}

async function output() {
   console.log('two');
   const req = new Request(url);
   const rep = await fetch(req);
   const json = await rep.json();
   console.log('five');
   viewData(json);
   //return await Promise.resolve('Hello');
}

function viewData(val) {
   console.log(val);
}

async function outputTest() {
   try {
       let rep = await Promise.resolve('Hello');
       if(!rep.ok){
```

```
            throw new Error('Error oh no');
        }
    } catch(error){
        console.log(error);
    }
}
```

AJAX and JSON in JavaScript

Web Page Data with AJAX from JSON file use AJAX JavaScript code

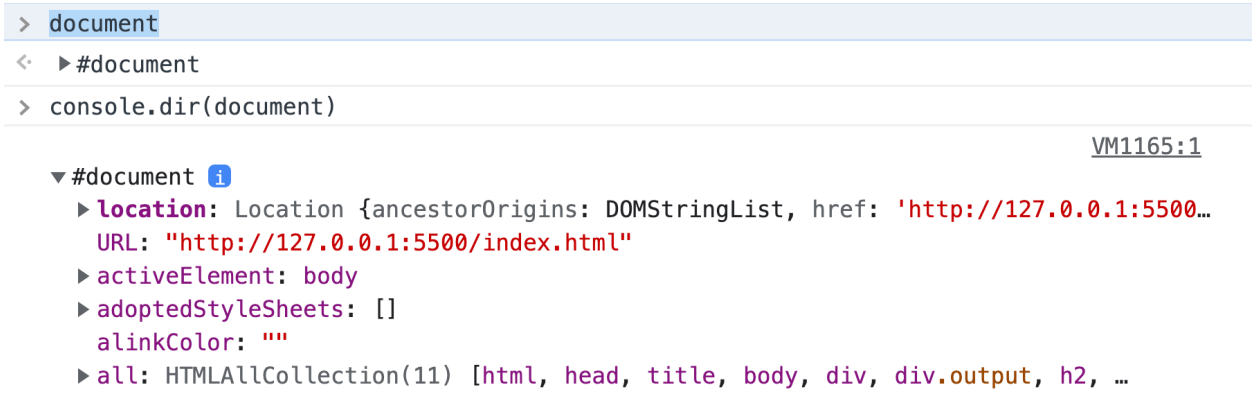
AJAX (Asynchronous JavaScript and XML) is not a technology but an approach to using several technologies together to create a user experience on a website. AJAX can create a better and more responsive customized experience to the user's actions. Although the XML which is the last letter in the acronym is not commonly used anymore, as it has been replaced by JSON. JSON offers many advantages over XML and is commonly used as a structure for the data. AJAX stands for Asynchronous JavaScript And XML. The features of AJAX include making a request to the server without a page reload, and receiving the data from the server.

In the previous lesson we saw how we can retrieve data back from a server using JavaScript fetch, which is a preferred method of making the request to the server in modern coding, as previously we were only using the XMLHttpRequest object.

To select page elements in JavaScript we can use the document object. This is a large object created by the browser to represent the page elements. JavaScript can select the page elements using the document object, add event listeners for interactive content, and update page elements dynamically with code. When accessing the page elements it is referred to as the DOM.

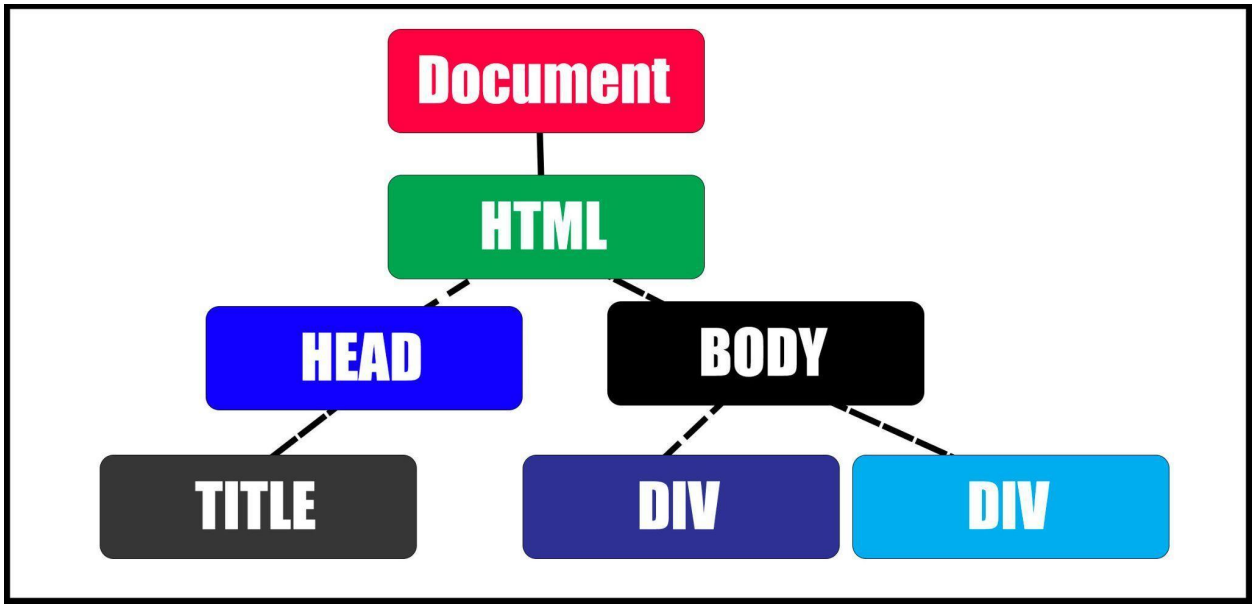
You can access the document of any web page in your browser console. Type document which will return the document back in the console. You can also type console.dir(document) to be able to open and view the properties and methods contained within the document object.

```
console.dir(document)
document
```



In your browser console typing the document will return the page document object.

The Document Object Model (DOM) is an object that contains a data representation of the page elements. The DOM is structured in a tree like format, as objects that comprise the web page and content of the HTML web document. Document Object Model (DOM) is a programming interface for HTML documents, that is the logical structure of a page and how the page content can be accessed and manipulated. Bring your web pages to life with JavaScript and connect to the web page elements. Accessing the DOM you can create fully interactive content that responds to the user. DOM and JavaScript lets you create Dynamic web page content that can change without page refresh and present new elements and updated content to the user. Improve your web users experience with JavaScript and the DOM.



To update the page element contents, you can assign new values to the properties. Select the element you want to update and then using the equal sign you can assign a new value to the page element. The document.body is the main body element from your web page. This can be selected and updated just like any other element in the document.

Go to any webpage, open your browser console and type the document.body.textContent = to your name and view the result.

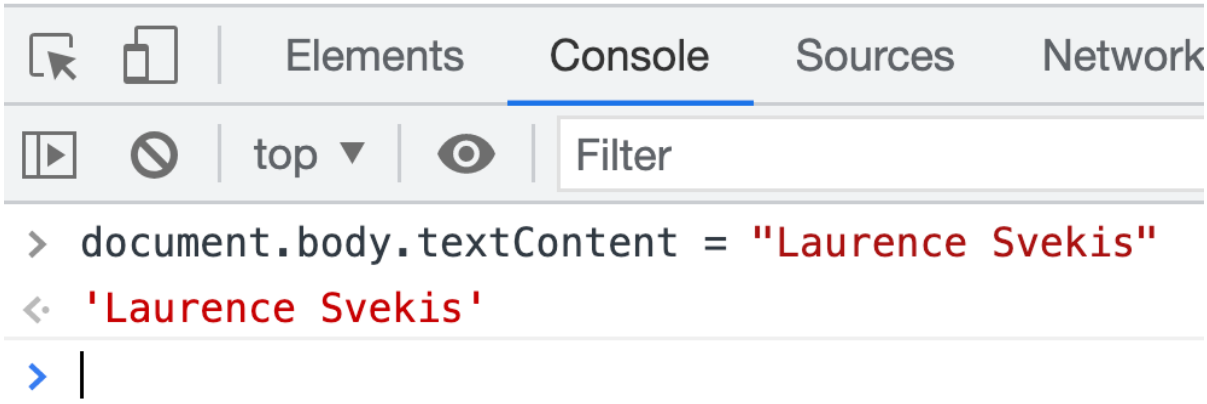
document.body.textContent = "Laurence Svekis"

The Document method querySelector() returns the first Element that is found within the document that matches the specified selector. If no results are found then the result that is returned is null.

There are many methods that can be used in JavaScript to select the element that you want to manipulate. Using document.querySelector() or document.querySelectorAll() can be used to select elements as you typically would using CSS. To select a tag use the tag name in the argument of the methods as a

string, to select a class prefix the class name with the dot, to select an element using its ID prefix it with a hash #.

Laurence Svekis



Selection and update of the property value for the body textContent

Elements in the web page can have events attached to them. When the action is taken the element can trigger a function to be invoked which will run a block of code on the action. There is an event object that is captured as well and can be used in the code to get the target element and other details about the event.

Events can be added using an addEventListener() method that sets up a function that gets called whenever the specified event is triggered on a target. You can also use a global event handler directly on the element such as onclick() which will trigger the function code once the event action is done. You can also use the addEventListener() method or the global event although the addEventListener() method is the recommended way to register an event listener.

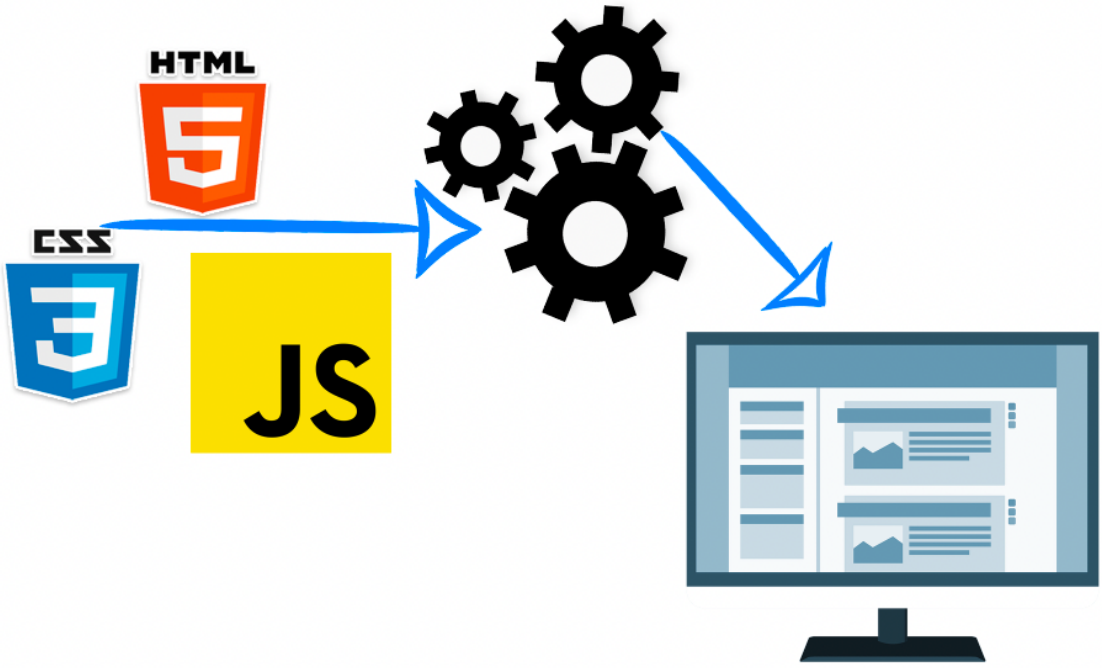
An event that can be listened for when the DOM content loads is the 'DOMContentLoaded' event. You can use this to run a block of code once the DOM elements are fully loaded and ready to be interacted with. If you try to interact with page elements before the DOM has loaded fully, you might not reach them

depending on the order that your code is set. If you add the JavaScript with the DOM interactions and selections before the body of your HTML, the elements won't have loaded yet and it will throw an error.

```
document.addEventListener('DOMContentLoaded', (e) => {
  getValues();
})
```

Once you have selected the element, you can update the property values of that element. output.innerHTML or output.textContent are a couple properties that can be used to update the visible values in the page content.

APIs- are sets of prebuilt building blocks that allow developers to connect into and access these objects. Think of it like a control panel which JavaScript language lets you interact with. The browser also has an API, which can be accessed by JavaScript code, and this is what allows for the interactions between the code and the visible web content. Browser APIs allow access to the web page elements, and other data. You can access the API and do useful programming things. The DOM (Document Object Model) API allows you to manipulate the HTML and CSS of the page. This book will cover the DOM in detail, as well as methods and ways that you can use JavaScript to select and manipulate page elements.



Exercise : Interactive web page with AJAX loading JSON data

1. Add several buttons to your HTML elements, as well add a div that can be used to output content to the user.
2. Using document.querySelector() select the page elements. document.querySelector() will select the first matching result. You can check the page element in the console using the variable.
3. Add an event listener to the first button, when pressed it should update the value of a counter variable.
4. Create a function to update the page element with the value of the counter, invoke this function every time the counter is updated.
5. Create an event that loads the value of the counter, using the JSON data from the json file. This should update the counter once it loads.
6. Add an event to the second button, once clicked it should make a request to the JSON file and update the counter with the JSON file value for counter.

INDEX.html

```
<!doctype html>
<html>

<head>
  <title>JavaScript JSON AJAX</title>
</head>

<body>
  <div>
     <div class="output"></div>
     <button class="btn">Click Me</button>
     <button class="btn1">Reset</button>
  </div>
  <script src="apps.js"></script>
</body>

</html>
```

APPS.JS

```
const btn = document.querySelector('.btn');
const btn1 = document.querySelector('.btn1');
const output = document.querySelector('.output');
let counter = 0;
const url = 'json.json';
console.log(btn);
output.innerHTML = '<h1>Hello World</h1>';

document.addEventListener('DOMContentLoaded', (e) => {
   getValues();
```

```
})

function getValues() {
    fetch(url)
        .then(rep => rep.json())
        .then(data => {
            console.log(data);
            counter = data.counter;
            updater();
        })
}

//output.textContent = '<h1>Hello World</h1>';

btn.onclick = (e) => {
    console.log(e.target);
    //adder();
}

btn.onclick = adder;
btn1.onclick = resetCounter;

function updater() {
    output.innerHTML = `<h2>Counter : ${counter}</h2>`;
}

function adder(e) {
    console.log(e.target);
```

```
    counter++;
    updater();
}

function resetCounter(e) {
    getValues();
}
```

JSON.JSON

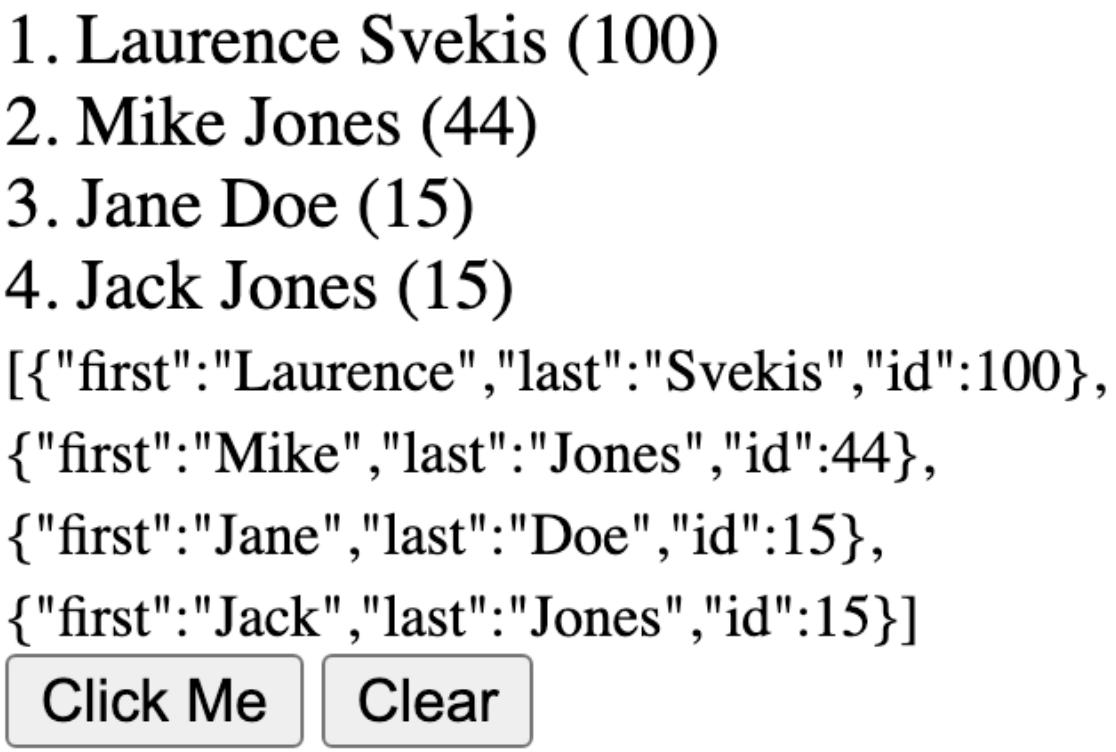
```
{
    "counter" : 25
}
```

Output JSON data into the web page

JavaScript code and JSON data for web page content. Use AJAX web development

AJAX communicates with the server and the exchange of data then updates the page without having to refresh the page. Features of AJAX allow you to make requests to the server without reloading the page and use received data from the server.

This lesson is going to demonstrate how to loop through the contents of an array, coming from JSON data and output the results into the web page.



JavaScript application to request JSON data and output the results to the webpage.

Exercise : Complex Data output

1. Create a JSON data file, with objects as items within an array.
2. Select the page elements from the HTML file that will be used in the code

3. Add an event listener to the first element that will invoke the fetch to the JSON data.
4. Add an event to the second button that will clear the contents of the output element.
5. Using the fetch method, make a request to the JSON file, return the data as a JavaScript object. Select the array that contains the items you want to output on the page.
6. Once the data is returned, loop through the contents of the array and output the data values into the web page in the output element.

JSON data :

[{"first":"Laurence","last":"Svekis","id":100},{"first":"Mike","last":"Jones","id":44},{"first":"Jane","last":"Doe","id":15},{"first":"Jack","last":"Jones","id":15}]

HTML

```
<!doctype html>
<html>
<head>
  <title>JavaScript JSON AJAX</title>
</head>
<body>
  <div>
     <div class="output"></div>
     <button class="btn">Click Me</button>
     <button class="btn1">Clear</button>
  </div>
  <script src="apps9.js"></script>
</body>
</html>
```

JAVASCRIPT

```
const btn = document.querySelector('.btn');
const btn1 = document.querySelector('.btn1');
const output = document.querySelector('.output');
const url = 'json6.json';
btn.onclick = () => {
   output.innerHTML = 'Connecting.....';
   getData();

}
btn1.onclick = () => {
   output.innerHTML = 'Cleared!';
}

function getData() {
   fetch(url)
       .then(rep => rep.json())
       .then(data => {
           outData(data.friends);
       })
}

function outData(val) {
   console.log(val);
   let html = '';
   val.forEach((ele, ind) => {
       console.log(ele);
```

```
        html += `<div>${ind+1}. ${ele.first} ${ele.last}
(${ele.id})</div>`;

    })
    html += `<small>${JSON.stringify(val)}</small>`;
    output.innerHTML = html;
}
```

Output JSON data into the WebPage

Using JavaScript how to connect to a web API endpoint for JSON data.

Get JSON data and dynamically generate page content with JavaScript. When connecting to server endpoints, a vast amount of data can then be used with your web application, allowing users to get data coming from those endpoints.

Web API's (Application Programming Interfaces) that fetch data from a server and return the data as JSON are commonly used. APIs can provide data for your web users. There are several API's that you can connect to, to practice making fetch requests. Many do require oAuth or other authenticate in order to be able to request it from your application. There are however some open APIs that allow you to connect to them with just the URL. The endpoints for these APIs can and do change, the setup and output is determined by the server application that creates the response object. Often the response is done as JSON which is a universally easy data structure for coding languages to parse. Some endpoints will have the parameters in the GET request URI, these can then be updated by updating the URL path including the new parameters.

Browsers load files via the HTTP or HTTPS protocol. HTTP is a stateless protocol which means once the content is served it won't remember anything about this request. It simply returns the request content, including parameters that are requested at the time of the request. Parameters can be included depending on if the server endpoint is able to receive them, by adding the parameters in the URL after the question mark. ? = parameters in URL

Headers can also be passed in the request, headers are determined by what the server requires. Headers provide additional information about content you expect back. Not all servers are open and many will change the status to avoid abuse of the requests.

Exercise - Retrieve complex data from an external API endpoint

1. Get the URL that you want to connect to. Check it within the browser to ensure that JSON data is returned. This example will use 'https://www.reddit.com/r/funny/top/.json?limit=5'
2. Select the page elements and add a click event to the page button.
3. Once the button is clicked it should make a fetch request to the data endpoint.
4. Output the results into the console, and select the array to loop through
5. Output the contents of the array into the webpage.

HTML

```
<!doctype html>
<html>
<head>
  <title>JavaScript JSON AJAX</title>
</head>
<body>
  <div>
     <div class="output"></div>
     <input type="number" value="5" min=0 max=10>
     <button class="btn">Click Me</button>
     <button class="btn1">Clear</button>
  </div>
  <script src="apps10.js"></script>
</body>
</html>
```

JAVASCRIPT

```
const btn = document.querySelector('.btn');
const btn1 = document.querySelector('.btn1');
```

```
const output = document.querySelector('.output');

const url = 'https://www.reddit.com/r/funny/top/.json?limit=5';

btn.onclick = (e)=>{
    fetch(url)
    .then(rep => rep.json())
    .then(data =>{
        getData(data.data.children);
    })
}

function getData(data){
    console.log(data);
    let html = '';
    data.forEach((item)=>{
        console.log(item.data);
        const el = item.data;
        html += `<div><h3>${el.title}</h3>`;
        html += `<img src='${el.thumbnail}'>`;
        html += `<a href='${el.url} target = `;html += `'_blank'
>${el.url}</a></div>`;
    })
    output.innerHTML = html;
}
```

GitHub API AJAX

AJAX endpoint connection for practice exploring the GitHub open API and get JSON data.

There are several open APIs endpoints that can be used to connect and retrieve data from. Testing these can be an excellent source for practice when connecting to large data JSON files. The navigation and selection of data that is deep within several levels of objects and arrays can be done piece by piece. Output the objects and arrays, then select the properties and values that you want to use in your code.

Connecting to external APIs that are open is not suggested for web development on live sites. These tend to change endpoints and once you build your connection your web page will be dependent on the connection access. There are many APIs that can be connected to the have authentication. For server side coding the API endpoints can be rendered by any backend code. The backend code would be outputting the JSON data and creating the dynamic outputs. Many endpoints are based on data coming from databases, the database connections and outputs are coded with the backend code. The server code will also control header access, so if you encounter an error this is one of the possible reasons.

Within the document object you can also create page elements as objects, this is an alternative to using the static innerHTML and then creating the elements with the HTML code. document.createElement('button') will create a button, which can then be updated. Since this is already an object, properties like the event listeners can be added to it. Once the element is created it needs to be added to the web page, to a parent element. This can be done with append() or prepend().

Exercise : Connect and output data from a large complex JSON data response

1. Select the page elements as variables within your JavaScript code
2. Add a click event then connect to the endpoint https://api.github.com/zen and return the text content outputting it into the web page.

3. On the second button connect to the endpoint https://api.github.com/ and output the object key and values. Create links to the URLs from the JSON data.
4. Use JavaScript to create a new page element button. document.createElement('button');
5. Add an event listener to the new button. Click of the button should invoke a function that connects to https://api.github.com/search/repositories?q=javascript then create a separate function to handle the output of the content into the webpage.
6. In the outputItems function get the array of data items. Loop through the data.items and build the HTML content, get the name, url, description and topics from the items and output them on the page.
7. Update the URL that is requested, with the value from the input field. This should change the number of items that are returned and output to the page.

HTML

```
<!doctype html>
<html>
<head>
  <title>JavaScript JSON AJAX</title>
</head>
<body>
  <div>
     <div class="output"></div>
     <input type="number" value="5" min=0 max=10>
     <button class="btn">Click Me</button>
     <button class="btn1">Clear</button>
  </div>
  <script src="apps11.js"></script>
</body>
```

```
</html>
```

JAVASCRIPT

```
const btn = document.querySelector('.btn');
const btn1 = document.querySelector('.btn1');
const btn2 = document.createElement('button');
const myInput = document.querySelector('input');
btn2.textContent = "load search results";
document.body.prepend(btn2);
const output = document.querySelector('.output');
btn.textContent = 'Get Zen Quote';
btn1.textContent = 'Endpoints';
const url = 'https://api.github.com/zen';
const url1 = 'https://api.github.com/';
const url2 =
'https://api.github.com/search/repositories?q=javascript';

btn.onclick = () => {
   fetch(url)
       .then(res => res.text())
       .then(zen => {
           console.log(zen);
           output.innerHTML = `<h1>${zen}</h1>`;
       })
}

btn1.onclick = () => {
   fetch(url1)
```

```
        .then(res => res.json())
        .then(data => {
            console.log(data);
            let html = '<ul>';
            for (const [key, value] of Object.entries(data)) {
                console.log(key, value);
                html += `<li><a href="${value}"
target="_blank">${key}</a></li>`;
            }
            html += `</ul>`;
            output.innerHTML = html;
        })
}

btn2.onclick = () => {
    const val = myInput.value;
    const main = `${url2}&per_page=${val}`;
    fetch(main)
        .then(res => res.json())
        .then(data => {
            output.innerHTML = `<div>Search for ${val} results
total:${data.total_count}</div>`;
            outputItems(data.items);
        })
}

function outputItems(data) {
    let html = '';
```

```
    data.forEach(el => {
        html += `<div style="border:1px solid #ddd">`;
        html += `<div>${el.name}</div>`;
        html += `<div>${el.url}</div>`;
        html += `<div>${el.description}</div>`;
        html += `<div>`;
        el.topics.forEach(topic => {
            html += `<span>${topic}</span> | `;
        })
        html += `</div></div>`;
    })
    output.innerHTML += html;
}
```

Stackoverflow API connection example

The stackoverflow API provides a massive JSON object that can be used to get content within your webpage. Please note that there are limits to the open API requests. More details about the API are at https://api.stackexchange.com/docs

Exercise : Get object data and dynamically create web page elements with code

1. Select the page elements, and create a variable for the URL path to connect to.
2. Using the btn onclick event, connect to the API with fetch.
3. Once data is retrieved, output the array of items into the web page. Create a separate function to handle the element creation and adding to the page.
4. Loop through all the items in the data, using document.createElement() create page elements, append them to the element with the class of output.

HTML

```
<!doctype html>
<html>
<head>
  <title>JavaScript JSON AJAX</title>
</head>
<body>
  <div>
     <div class="output"></div>
     <button class="btn">Click Me</button>
  </div>
  <script src="apps12.js"></script>
</body>
</html>
```

JAVASCRIPT

```
const btn = document.querySelector('.btn');
const btn1 = document.querySelector('.btn1');
const myInput = document.querySelector('input');
const output = document.querySelector('.output');
const base = 'https://api.stackexchange.com/';
const pathURL =
'2.3/questions?order=desc&sort=activity&site=stackoverflow';

btn.onclick = (e)=>{
   const url = base + pathURL;
   fetch(url)
   .then(rep => rep.json())
   .then(data =>{
       console.log(data);
       outputData(data.items);
   })
}

function outputData(data){
   console.log(data);
   data.forEach((el) =>{
       console.log(el);
       const main = document.createElement('div');
       output.append(main);
       const div1 = document.createElement('div');
       div1.textContent = el.title;
       main.append(div1);
```

```
        const div2 = document.createElement('div');
        main.append(div2);
        el.tags.forEach((tag)=>{
            const span = document.createElement('span');
            span.textContent = tag;
            div2.append(span);
        })
        const div3 = document.createElement('div');
        main.append(div3);
        div3.innerHTML = `<a href="${el.link}"
target="_blank">${el.link}</a>`;
    })
}
```