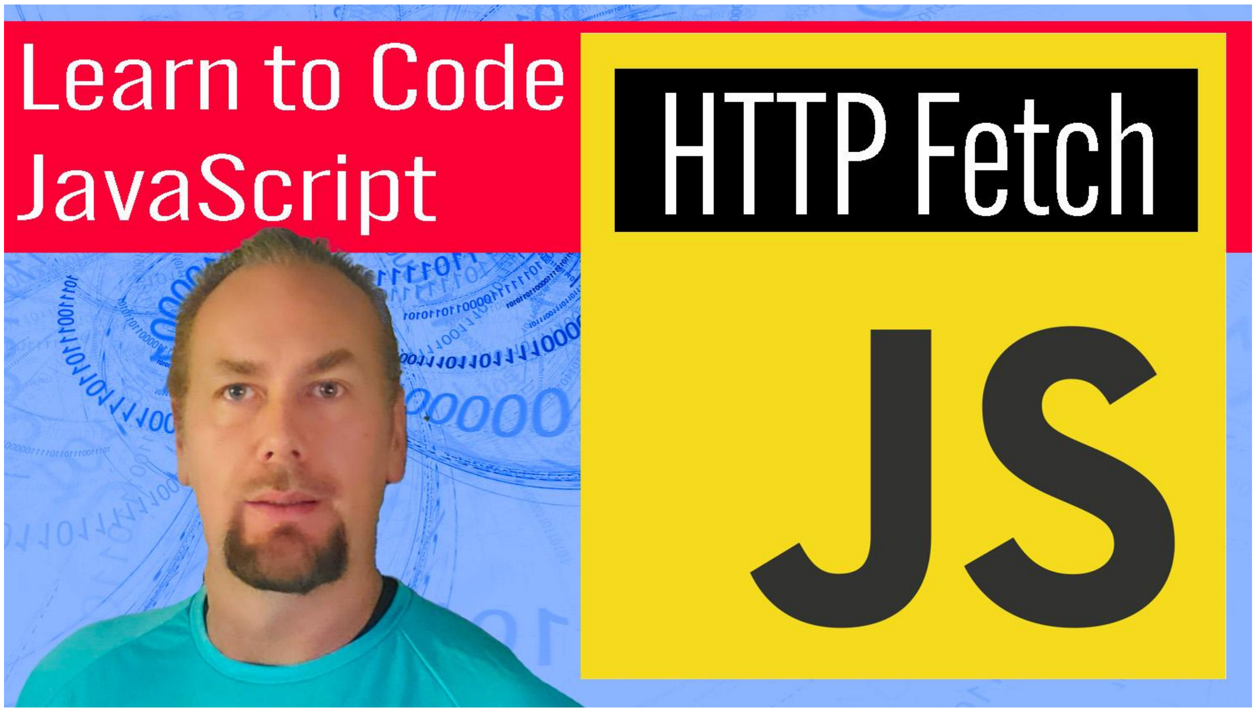
Sample JavaScript Code Laurence Svekis



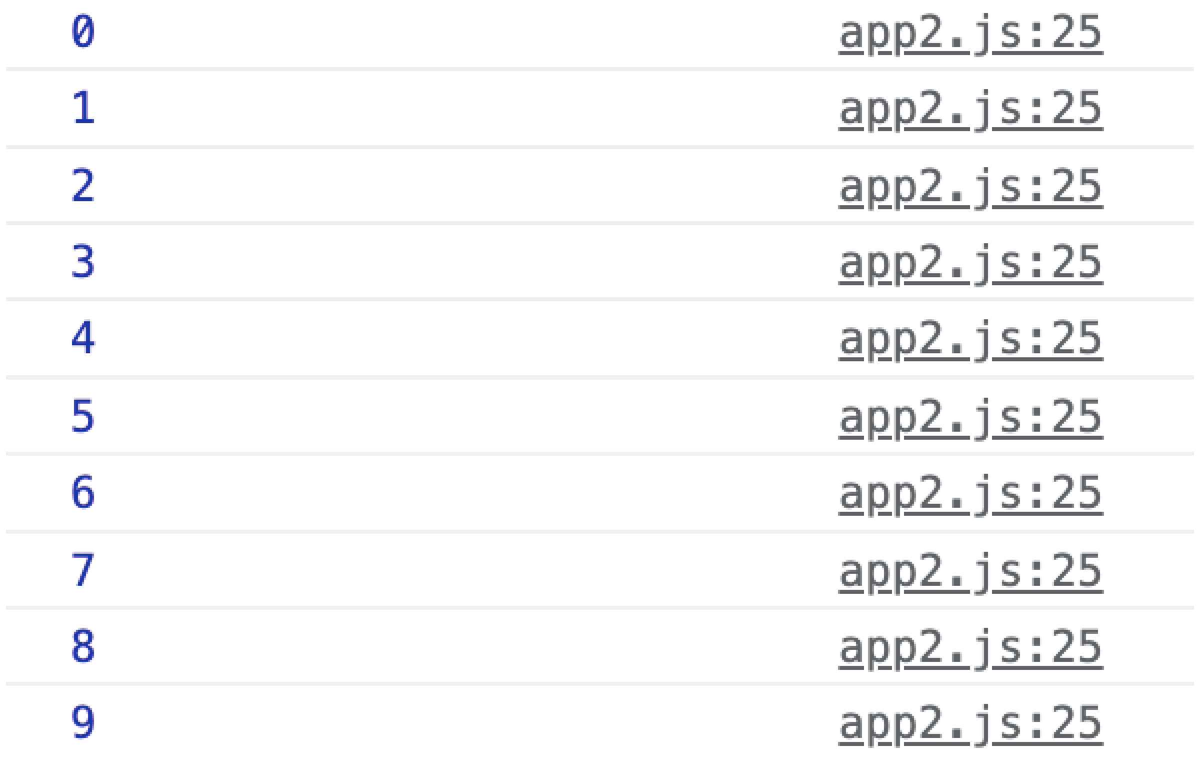
JavaScript Closure

A closure in JavaScript is a function that has access to the variables in its parent scope, even after the parent function has completed execution. This allows for data to be "closed over" or remembered by the inner function, even after the outer function has returned.

https://youtu.be/AyQRYwV69cc

```
For example:
function makeCounter() {
  let count = 0;
  return function() {
    return count++;
  }
}

let counter = makeCounter();
console.log(counter()); // outputs 0
console.log(counter()); // outputs 1
console.log(counter()); // outputs 2
```

Here, the makeCounter function returns an inner function that has access to the count variable declared in its parent scope, and can "remember" the current count value even after the makeCounter function has completed execution. Each time the inner function is called, it returns the current value of count and increments it by 1.



```
0                                  app2.js:25
1                                  app2.js:25
2                                  app2.js:25
3                                  app2.js:25
4                                  app2.js:25
5                                  app2.js:25
6                                  app2.js:25
7                                  app2.js:25
8                                  app2.js:25
9                                  app2.js:25
```

```
const a = 'hello';
```

```
console.log(a);
abc();

function abc(){
   //const a = 'world';
   console.log(a);
}

function myCount(){
   let count = 0;
   return function(){
       return count++;
   }
}
function myCount2(){
   let count = 0 ;
   return count++;
}

let cnt = myCount();
let cnt2 = myCount2;

for(let x=0;x<10;x++){
   console.log(cnt());
```

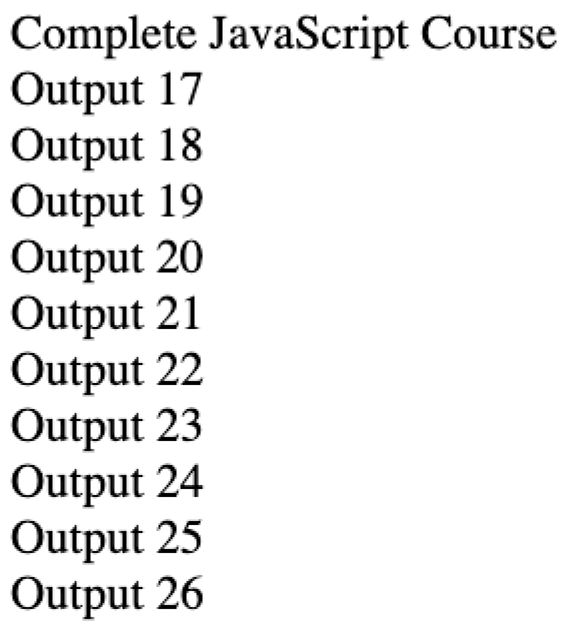
```
    console.log(cnt2());
}
```

JavaScript Closure Advanced

In this example, the makeAdder function takes in a single argument x and returns an inner function that takes in a second argument y. The inner function has access to the x variable declared in the parent scope and uses it to add x and y together and return the result.
We can see here that the outer function makeAdder has been executed twice and it returns two different inner functions which are assigned to different variables add5 and add10 and these inner functions are able to remember their respective parent scope values of x.

https://youtu.be/8EgbirmLt0g

```
function makeAdder(x) {
  return function(y) {
    return x + y;
  }
}

let add5 = makeAdder(5);
console.log(add5(3)); // outputs 8
console.log(add5(4)); // outputs 9

let add10 = makeAdder(10);
console.log(add10(5)); // outputs 15
console.log(add10(6)); // outputs 16
```



```
Complete JavaScript Course
Output 17
Output 18
Output 19
Output 20
Output 21
Output 22
Output 23
Output 24
Output 25
Output 26
```

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

function adder(val){
   return function(val2){
       return val + val2;
   }
}

let a1 = adder(15);
console.log(a1(2));

for(let x=0;x<10;x++){
  output.innerHTML += `<div>Output ${(a1(2+x))}</div>`;
}
```

JavaScript Image Gallery and Dynamic Image Gallery using page classes or create page elements on the fly with code

https://youtu.be/nsGGMAYnLbs

Here is an example of a JavaScript image gallery maker that creates a simple image gallery with prev/next buttons to navigate through the images:

```
<div id="gallery">
  <img src="image1.jpg" id="current-image">
  <button id="prev-button">Prev</button>
  <button id="next-button">Next</button>
</div>

<script>
  var images = ["image1.jpg", "image2.jpg",
"image3.jpg", "image4.jpg"];
  var currentIndex = 0;

  var gallery = document.getElementById("gallery");
  var currentImage =
document.getElementById("current-image");
  var prevButton =
document.getElementById("prev-button");
```

```
  var nextButton =
document.getElementById("next-button");

  prevButton.addEventListener("click", function() {
    currentIndex--;
    if (currentIndex < 0) {
      currentIndex = images.length - 1;
    }
    currentImage.src = images[currentIndex];
  });

  nextButton.addEventListener("click", function() {
    currentIndex++;
    if (currentIndex >= images.length) {
      currentIndex = 0;
    }
    currentImage.src = images[currentIndex];
  });
</script>
```

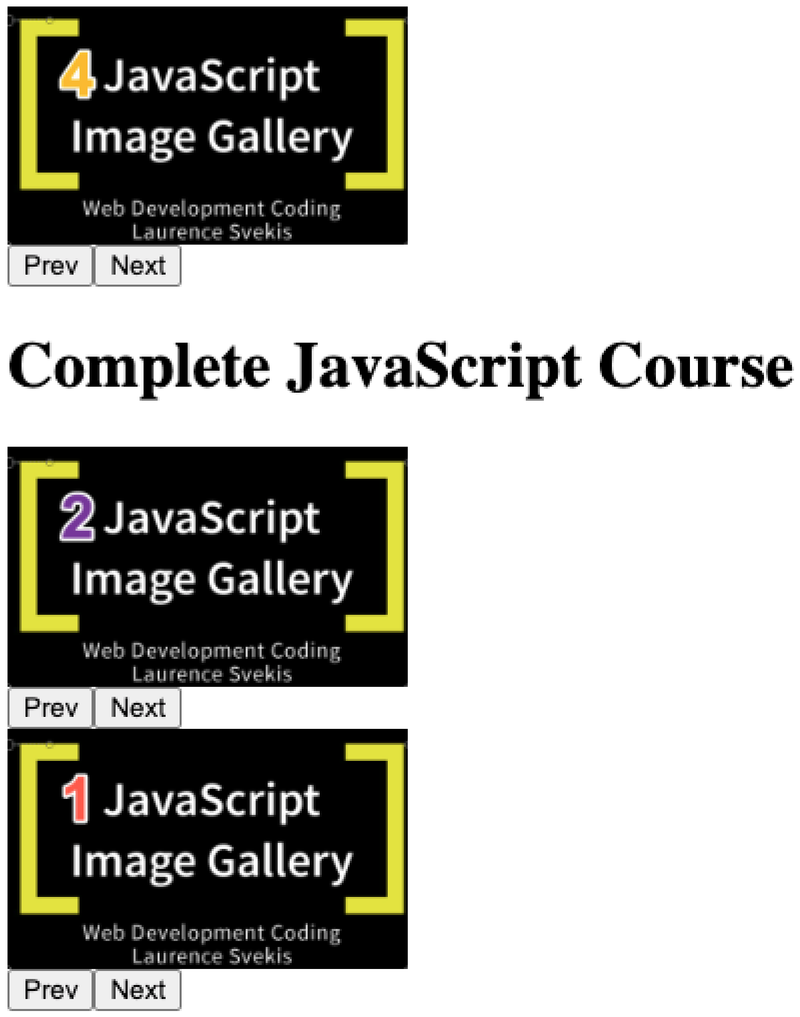
This example uses JavaScript to select the elements from the HTML, and add event listeners to the prev/next buttons to navigate through the images in the images array when clicked. The currentIndex variable keeps track of the current image being displayed, and the currentImage.src property is updated to show the next/prev image in the array when the buttons are clicked.

The above code is an example of a JavaScript image gallery maker that creates a simple image gallery with prev/next buttons. The code uses JavaScript to select the necessary elements from the HTML, such as the gallery container, current image, and prev/next buttons. It then adds event listeners to the prev/next buttons, so that when they are clicked, the current image being displayed is updated to the next/prev image in the images array. The currentIndex variable keeps track of the current image being displayed, and it is updated each time the prev/next buttons are clicked. When the current index reaches the end of the images array, it resets to the first image, thus creating an infinite loop.

Dynamic Image Gallery

How to Create an Image Gallery and Dynamic Image Gallery with JavaScript Code

The image gallery can also be used within a function to create multiple image galleries all working independently. Either creating them on the fly within the code or selecting existing elements with the class name and generating images within those elements.



Complete JavaScript Course

```
const output = document.querySelector('.output');
const images =
['one.jpg','two.jpg','three.jpg','four.jpg'];
/*
for(let x=0;x<12;x++){
   const el = document.createElement('div');
   output.append(el);
   cGallery(el);
}
```

```
*/
const eles = document.querySelectorAll('.gal');
eles.forEach(el => {
   cGallery(el);
})

function cGallery(parentEle){
   let curIndex = 0;
   const gallery = document.createElement('div');
   const curImage = document.createElement('img');
   curImage.setAttribute('src','one.jpg');
   const btn1 = document.createElement('button');
   btn1.textContent = 'Prev';
   const btn2 = document.createElement('button');
   btn2.textContent = 'Next';
   parentEle.append(gallery);
   gallery.append(curImage);
   gallery.append(btn1);
   gallery.append(btn2);

   btn1.addEventListener('click',()=>{
       curIndex--;
       if(curIndex<0){
           curIndex = images.length-1;
```

```
        }
        console.log(images[curIndex]);
        curImage.src = images[curIndex];
    })
    btn2.addEventListener('click',()=>{
        curIndex++;
        if(curIndex >= images.length){
            curIndex = 0;
        }
        console.log(images[curIndex]);
        curImage.src = images[curIndex];
    })
}
```

HTTP request in Javascript Get JSON data with xhr method and fetch methods

HTTP request in Javascript?

There are several ways to make an HTTP request in JavaScript, including using the XMLHttpRequest object or the fetch() function.

Here is an example of making an HTTP GET request using XMLHttpRequest:

```
var xhr = new XMLHttpRequest();
xhr.open("GET", "https://example.com");
xhr.send();
```

Here's an example of making an HTTP GET request to a JSON endpoint using the XMLHttpRequest object and parsing the response as JSON:

```
var xhr = new XMLHttpRequest();
xhr.open("GET", "https://example.com/data.json");
xhr.onload = function() {
    if (xhr.status === 200) {
        var data = JSON.parse(xhr.responseText);
        console.log(data);
    } else {
        console.error(xhr.statusText);
    }
};
xhr.onerror = function() {
    console.error(xhr.statusText);
};
xhr.send();
```

In this example, a new XMLHttpRequest object is created, and then opened with the "GET" method and the specified JSON endpoint URL. The onload event is used to handle the response, and onerror event is used to handle any error. The xhr.status is checked and if it's 200, it indicates that the request is successful, then we parse the response as JSON and

log the data to the console. If the xhr.status is not 200, it means there's an error and it logs the error message in the onerror function. If there's any network error, the onerror function is triggered, and it logs the error message.
Finally the request is sent using the xhr.send() method.

Please note that you should always check the response status code and handle it accordingly. Also, XMLHttpRequest is an old API, and **fetch() is more modern and recommended**

The fetch() method is a modern JavaScript method that allows you to make HTTP requests, similar to the XMLHttpRequest object. The fetch() method returns a promise that resolves to the response of the request, which can be a Response or Error object.

When you call the fetch() method, you pass in the URL of the endpoint you want to make the request to. You can also pass in an options object as the second parameter, which allows you to configure the request, such as setting the HTTP method, headers, and body.

The fetch() method returns a promise that resolves to the response of the request. Once you have the response, you can use the .json(), .text(), .blob() methods, etc to access the data of the response.

Here's an example of how you can use the fetch() method:

```
fetch("https://example.com/data.json")
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.log(error));
```

In this example, the fetch() method is used to make a GET request to a JSON endpoint. The .then() method is used to handle the response, which is passed as a parameter to the first callback function. The response.json() method is used to parse the response as JSON and the result is passed to the second callback function. Finally, the data is logged to the console. If there's any error during the request, it will be caught and logged by the catch function.

The fetch() method is a more modern and recommended way to make HTTP requests in JavaScript, it's more concise and easy to use, and it's supported in most modern browsers.

And here is an example of making an HTTP GET request using fetch():

```
fetch("https://example.com")
  .then(response => response.text())
  .then(data => console.log(data))
  .catch(error => console.log(error));
```

The first example uses the XMLHttpRequest object to create a new request, open it with the "GET" method and the specified URL, and then send it. The response to the request can then be handled using the onload or onerror events.

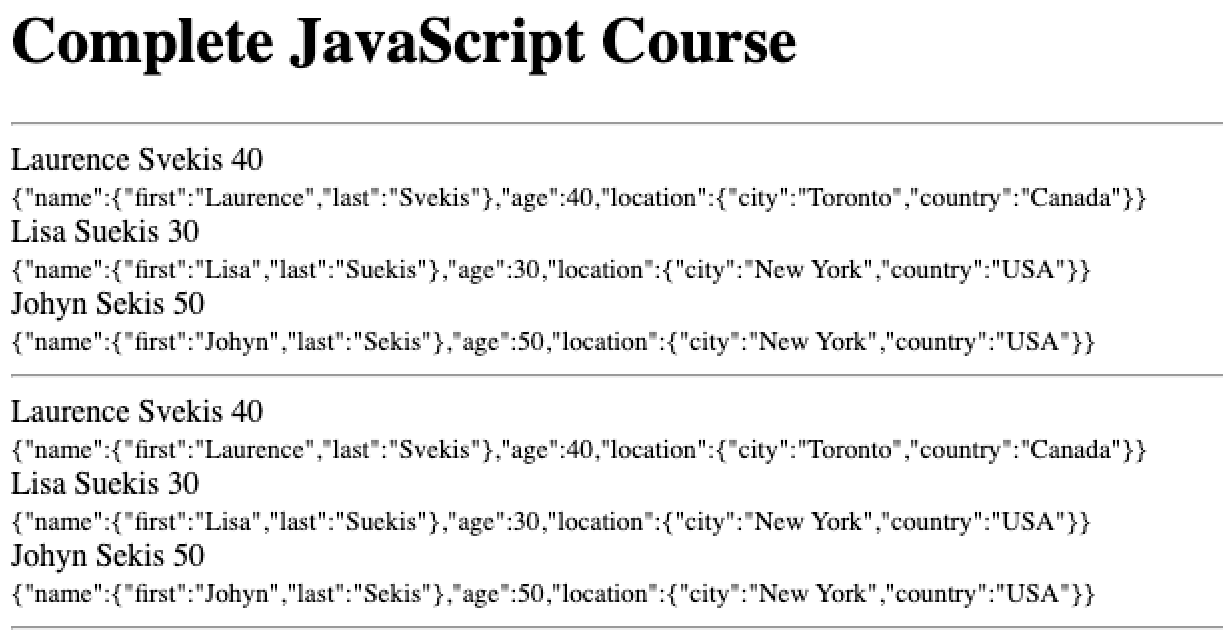
The second example uses the fetch() function to make the same GET request to the specified URL, and then uses the .then() method to handle the response, which is passed as a parameter to the first callback function. The response is transformed to text and then logged in the second callback function. If there's any error during the request, it will be caught and logged by the catch function.

Here's an example of making an HTTP GET request to a JSON endpoint using the fetch() function and parsing the response as JSON:

```
fetch("https://example.com/data.json")
```

```
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.log(error));
```

The fetch() function is used to make the GET request to the specified JSON endpoint. The response.json() method is then used to parse the response as JSON and the result is passed to the first callback function. In the second callback function, the data is logged. If there's any error during the request, it will be caught and logged by the catch function.



```
Complete JavaScript Course

Laurence Svekis 40
{"name":{"first":"Laurence","last":"Svekis"},"age":40,"location":{"city":"Toronto","country":"Canada"}}
Lisa Suekis 30
{"name":{"first":"Lisa","last":"Suekis"},"age":30,"location":{"city":"New York","country":"USA"}}
Johyn Sekis 50
{"name":{"first":"Johyn","last":"Sekis"},"age":50,"location":{"city":"New York","country":"USA"}}

Laurence Svekis 40
{"name":{"first":"Laurence","last":"Svekis"},"age":40,"location":{"city":"Toronto","country":"Canada"}}
Lisa Suekis 30
{"name":{"first":"Lisa","last":"Suekis"},"age":30,"location":{"city":"New York","country":"USA"}}
Johyn Sekis 50
{"name":{"first":"Johyn","last":"Sekis"},"age":50,"location":{"city":"New York","country":"USA"}}
```

```
const output = document.querySelector('.output');
const url =
'https://www.discoveryvip.com/shared/person1000.json';
const xhr = new XMLHttpRequest();
xhr.open('GET',url);
xhr.onload = function(){
   if(xhr.status === 200){
       const data = JSON.parse(xhr.responseText);
       maker(data);
   }else{
       console.error(xhr.statusText);
   }
}
xhr.onerror = function(){
   console.error(xhr.statusText);
}
xhr.send();
output.innerHTML += '<hr>';

fetch(url)
   .then(res => res.json())
   .then(data =>   maker(data))
   .catch(error => console.log(error));


function maker(data){
   data.forEach(ele =>{
       output.innerHTML += `
       <div>${ele.name.first} ${ele.name.last}
${ele.age}</div>
```

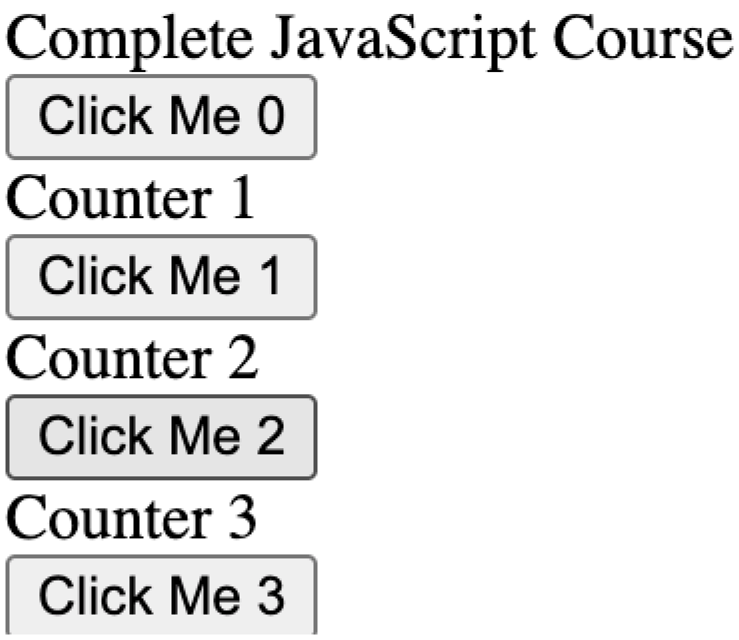
```
        <small>${JSON.stringify(ele)}</small>`;
    })
    output.innerHTML += '<hr>';
}
```

JSON Code

```
[
   {
     "name": {
       "first": "Laurence",
       "last": "Svekis"
     },
     "age": 40,
     "location": {
       "city": "Toronto",
       "country": "Canada"
     }
   },
   {
     "name": {
       "first": "Lisa",
       "last": "Suekis"
     },
     "age": 30,
     "location": {
       "city": "New York",
       "country": "USA"
     }
   },
   {
     "name": {
       "first": "Johyn",
       "last": "Sekis"
     },
     "age": 50,
     "location": {
       "city": "New York",
       "country": "USA"
     }
   }
 ]
```

How to add Fade Out and Fade in to page elements pure JavaScript

Learn how to apply fade in and fade out effects to HTML page elements with pure JavaScript code. Select and create new page elements dynamically with code, add event listeners and have the page elements fade in and fade out once the event is triggered.
Adding Fade Effects to new and existing Page Elements



Complete JavaScript Course

Counter 1

Counter 2

Counter 3

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

for(let x=0;x<5;x++){

   const el = document.createElement('div');

   output.append(el);



   const btn = document.createElement('button');

   btn.textContent = `Click Me ${x}`;

   el.append(btn);



   const div = document.createElement('div');
```

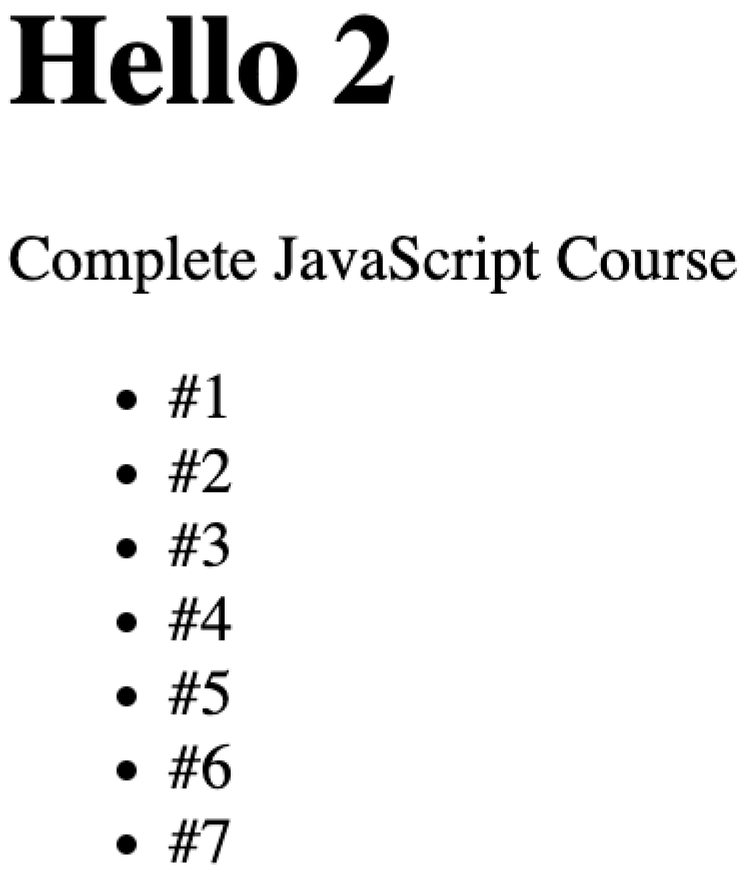
```
    div.style.transition = 'opacity 1500ms';

    div.style.opacity = '1';

    div.textContent = `Counter ${x+1}`;

    el.append(div);


    btn.addEventListener('click',()=>{

        if(div.style.opacity === '1'){

            div.style.opacity = '0';

        }else{

            div.style.opacity = '1';

        }

    })

}


const fademe = document.querySelectorAll('.fader');

fademe.forEach((ele)=>{

    ele.style.transition = 'opacity 500ms';
```

```
   ele.style.opacity = '1';

   ele.addEventListener('click',(e)=>{

       ele.style.opacity = '0';

   })

})

   <div id="output">Complete JavaScript Course </div>

   <div class="fader">One</div>

   <div class="fader">Two</div>

   <div class="fader">Three</div>

   <div class="fader">Four</div>

   <script src="app1.js"></script>
```

How to create page HTML elements with JavaScript code append prepend before after pure JavaScript

How to append and add new page elements with JavaScript
How to append and add new page elements with JavaScript using append, appendChild, prepend, before and after methods to dynamically add and reposition page elements

Create Page elements with Code
How to append and add new page elements with JavaScript



Hello 2

Complete JavaScript Course

- #1
- #2
- #3
- #4
- #5
- #6
- #7

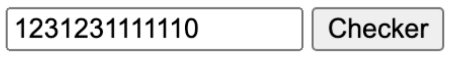
```
const output = document.querySelector('#output');

const pageBody = document.body;

const el1 = document.createElement('h1');

el1.textContent = 'Hello World 1';

console.log(el1);

pageBody.append(el1);

output.append(el1);

const res1 = output.appendChild(el1);

console.log(res1);


res1.textContent = 'Hello 1';

el1.textContent = 'Hello 2';

output.before(el1);

output.after(el1);

output.prepend(el1);

const ul = document.createElement('ul');

output.append(ul);
```

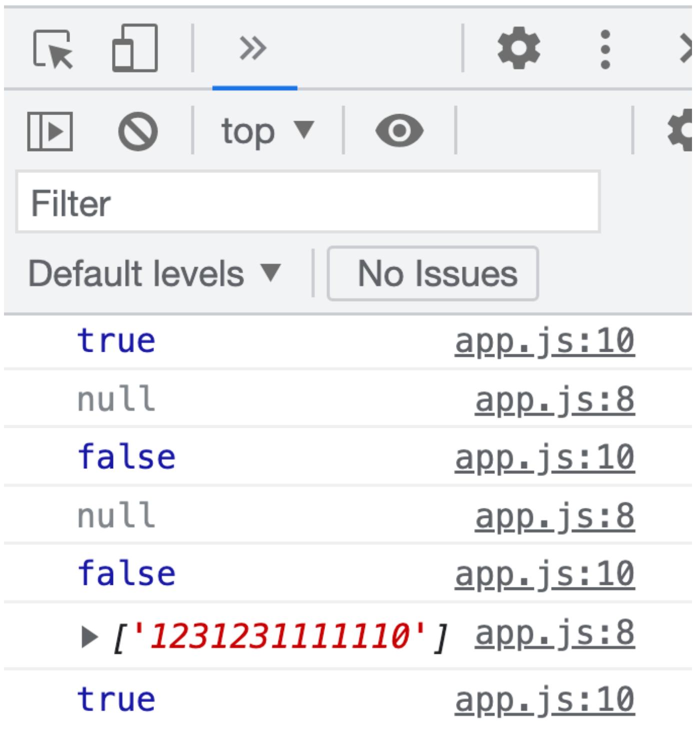
```
for(let i=0;i<10;i++){

   const li1 = document.createElement('li');

   li1.textContent = `#${i+1}`;

   ul.append(li1);

}
```

Regex Checking for Numbers in the input field

Check for values that match a Regex pattern in the input field. Push a button and apply the match checker to return the results in the console.

/^*[0-9]**$/g = Only numbers in the string
 /*[0-9]*+/g = Will return numbers in the result ignore non digits 0-9
/*[\D]*/g = Every Character other than digits
 /*d*/g = Digits separated



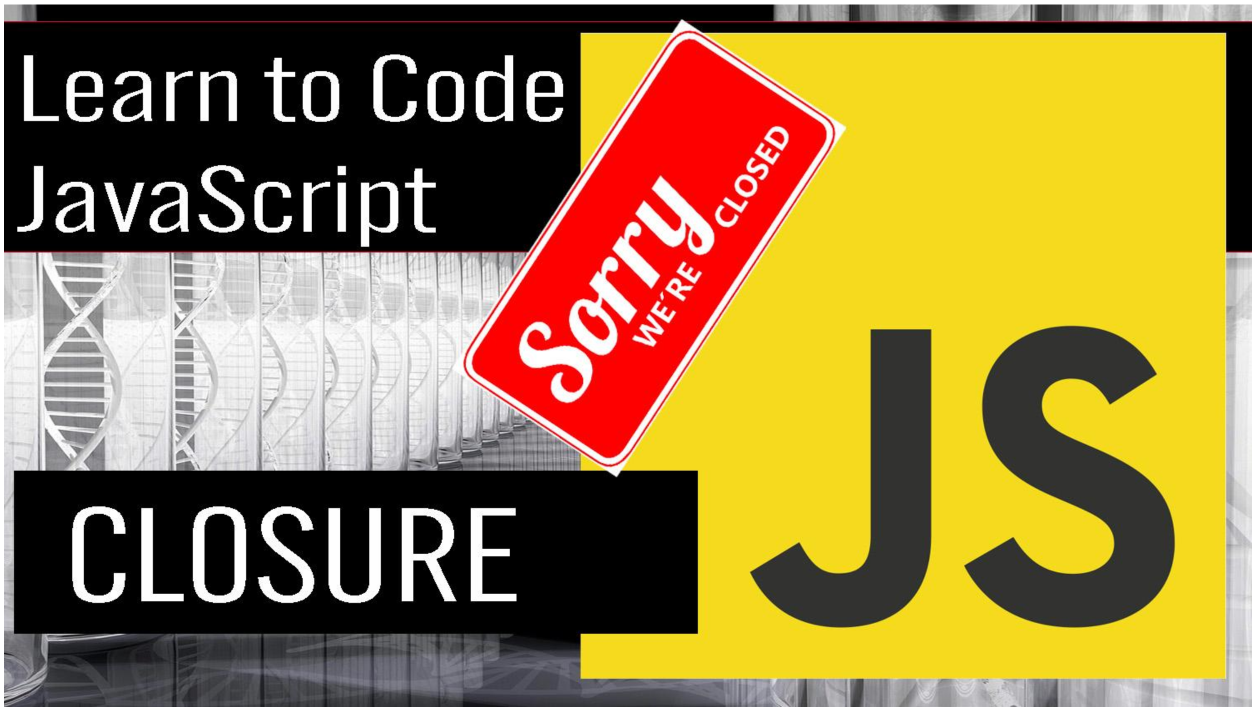


```
true                      app.js:10
null                      app.js:8
false                     app.js:10
null                      app.js:8
false                     app.js:10
▶ ['1231231111110']       app.js:8
true                      app.js:10
```

```
<!DOCTYPE html>
<html>
<head>
 <title>JavaScript Course</title>
</head>
<body>
 <div>
   <input type="text" id="nums">
   <button id="btn">Checker</button>
 </div>
 <script src="app.js"></script>
```

```
</body>
</html>
```

```
const nums = document.querySelector('#nums');
const btn = document.querySelector('#btn');

btn.onclick = ()=>{
   const inputValue = nums.value;
   const patt = /^[0-3]*$/g;
   const results = inputValue.match(patt);
   console.log(results);
   const valNum = results != null;
   console.log(valNum);
}
```



JavaScript Closure Explained

https://youtu.be/7vyHA0aODeU

A closure in JavaScript is a function that has access to variables in its parent scope, even after the parent function has returned. Closures are created when a function is defined inside another function, and the inner function retains access to the variables in the outer function's scope.

Here is an example of a closure in JavaScript:

code example
```
function outerFunction(x) {
    var innerVar = 4;
    function innerFunction() {
        return x + innerVar;
    }
    return innerFunction;
}

var closure = outerFunction(2);
console.log(closure()); // Output: 6
```

In this example, the innerFunction is a closure because it has access to the variable x and innerVar from the outerFunction even after outerFunction has returned.

A closure has three scope chains:
1. It has access to its own scope (variables defined between its curly braces {}).
2. It has access to the outer function's variables.

3. It has access to the global variables.

Closures are commonly used in JavaScript for a variety of tasks, such as:

- Implementing private methods and variables.
- Creating callback functions that retain access to variables from their parent scope.
- Creating and returning an object that has access to variables from its parent scope.

JavaScript closures are an important concept and it is important to understand how closures work in JavaScript. It is also important to be aware of the scope chain, and how closures interact with the scope chain.

```
<!DOCTYPE html>
<html>
<head>
  <title>Learn JavaScript</title>
</head>
<body>
  <h1>Complete JavaScript Course </h1>
  <div class="output"></div>
  <script src="app6.js"></script>
</body>
</html>
```

```
const val1 = 10;

function outerFun(x){
  const val2 = 10;
  function innerFun(){
        return x + val2 + val1;
    }
    return innerFun;
}

const val3 = outerFun(15);
console.log(val3());

for(let x=0;x<10;x++){
    console.log(outerFun(x+2)());
}
```

JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

https://youtu.be/wdoIV_09xAc

Here is an example of JSON data:

```
{
  "name": "Laurence Svekis",
  "age": 41,
  "address": {
     "street": "10 Main St",
     "city": "New York",
     "state": "NY",
     "zip": 10001
  },
  "phoneNumbers": [
     {
        "type": "home",
        "number": "212 123-1234"
     },
     {
        "type": "work",
      "number": "646 123-4567"
    }
  ]
}
```

JavaScript provides methods JSON.stringify() and JSON.parse() to convert between JSON and JavaScript objects.

Example of converting JavaScript object to JSON:

Code Example :

```
const object = { name: 'John Doe', age: 35 };
const json = JSON.stringify(object);
console.log(json);
Example of converting JSON to JavaScript object:
```

Code Example :

```
const json = '{"name":"John Doe","age":35}';
const object = JSON.parse(json);
console.log(object.name); // "John Doe"
```

In summary, JSON is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It is based on a subset of JavaScript and can be used with many programming languages. JavaScript provides built-in methods for converting between JSON and JavaScript objects.

There are several ways to get JSON data with JavaScript. One common method is to use the fetch() function to make an HTTP request to a server that returns JSON data. The fetch() function returns a promise that resolves to a response object, from which the JSON data can be extracted using the json() method.

Here is an example of how to get JSON data from a remote server:

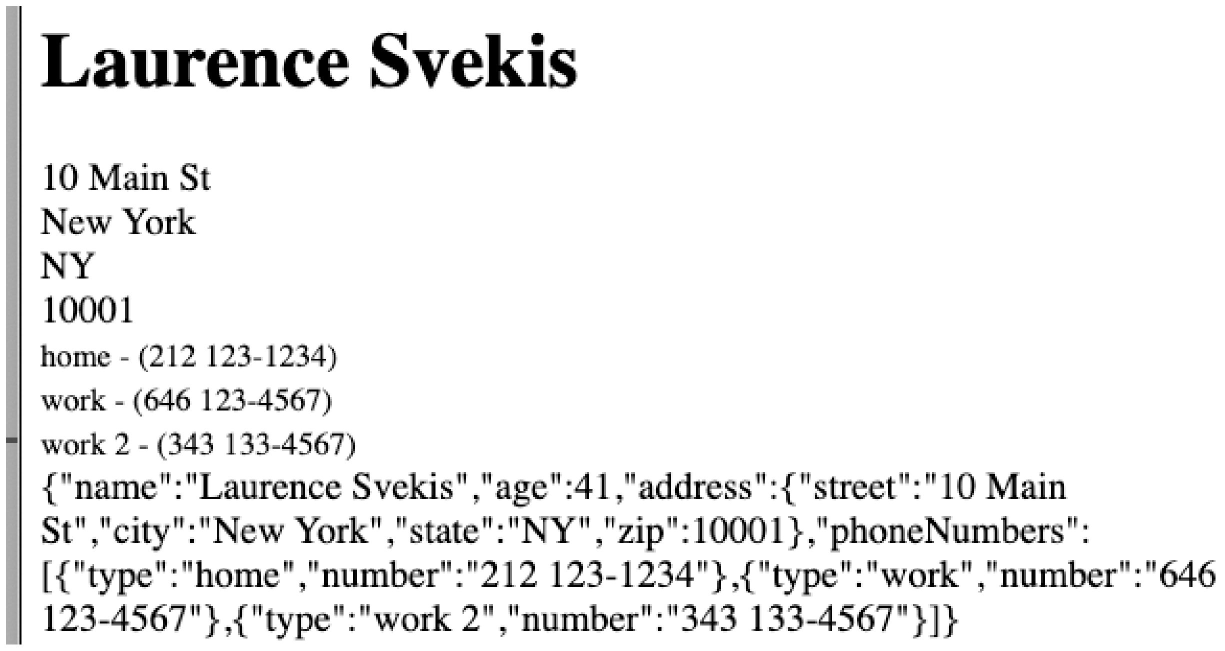
```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => {
    console.log(data);
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

Another way to get JSON data is to load it from a local file using the XMLHttpRequest object or the fetch() function.

Here is an example of how to get JSON data from a local file:

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'data.json', true);
xhr.responseType = 'json';
xhr.onload = function() {
  if (xhr.status === 200) {
    console.log(xhr.response);
  }
};
xhr.send();
```

In summary, there are several ways to get JSON data with JavaScript, including using the fetch() function to make an HTTP request to a server that returns JSON data or by loading JSON data from a local file using the XMLHttpRequest object or the fetch() function. Once you have the data you can use json() to access the data.



Laurence Svekis

```
10 Main St
New York
NY
10001
home - (212 123-1234)
work - (646 123-4567)
work 2 - (343 133-4567)
{"name":"Laurence Svekis","age":41,"address":{"street":"10 Main St","city":"New York","state":"NY","zip":10001},"phoneNumbers":[{"type":"home","number":"212 123-1234"},{"type":"work","number":"646 123-4567"},{"type":"work 2","number":"343 133-4567"}]}
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Learn JavaScript</title>
</head>
<body>
  <h1>Complete JavaScript Course </h1>
  <div class="output">Data</div>
  <script src="app7.js"></script>
</body>
</html>
```

```
const url = 'my1.json';
const output = document.querySelector('.output');
const dataSt = '{"name":"Laurence
Svekis","age":41,"address":{"street":"10 Main St","city":"New
York","state":"NY","zip":10001},"phoneNumbers":[{"type":"home","number
":"212 123-1234"},{"type":"work","number":"646
123-4567"},{"type":"work 2","number":"343 133-4567"}]}';
console.log(dataSt);
const dataObj = JSON.parse(dataSt);
console.log(dataObj);

output.addEventListener('click',getJsonData);

function getJsonData(){
   output.textContent = 'loading.....';
   fetch(url)
   .then(response => response.json())
   .then(data => {
       myOutput(data);
   })
   .catch(error => {
       console.error('Error:',error);
   })
}

function myOutput(data){
   let html = `<h1>${data.name}</h1>
<div>${data.address.street}</div>
<div>${data.address.city}</div>
<div>${data.address.state}</div>
<div>${data.address.zip}</div>
`;
data.phoneNumbers.forEach(el =>{
  html += `<small>${el.type} - (${el.number})</small><br>`;
})
html += JSON.stringify(data);
  output.innerHTML = html;
}
```

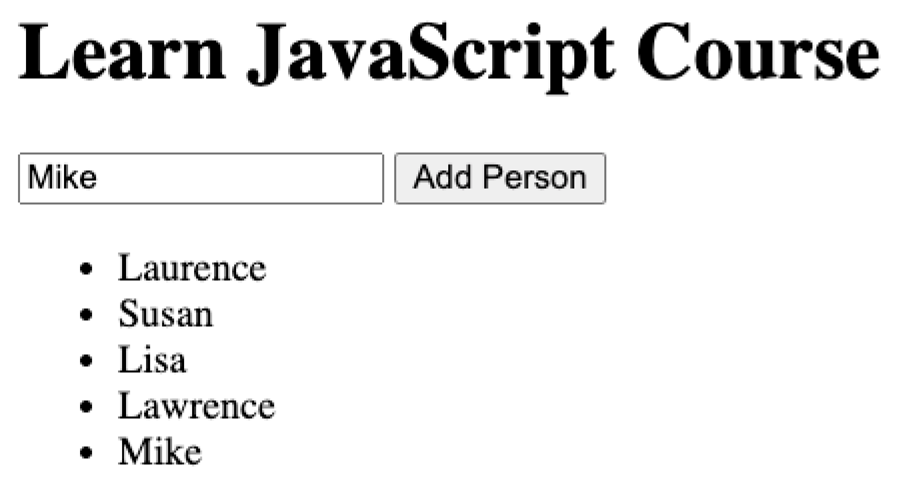
```
{
  "name": "Laurence Svekis",
  "age": 41,
  "address": {
     "street": "10 Main St",
     "city": "New York",
     "state": "NY",
     "zip": 10001
  },
  "phoneNumbers": [
     {
        "type": "home",
        "number": "212 123-1234"
     },
     {
        "type": "work",
        "number": "646 123-4567"
     },
     {
        "type": "work 2",
        "number": "343 133-4567"
      }
  ]
}
```

JavaScript Create Element List

https://youtu.be/oBuBoCrLRWg

The document.createElement() method in JavaScript is used to create a new HTML element with a specified tag name. The method takes a single argument, which is the tag name of the element to be created. For example, document.createElement("div") creates a new div element. The newly created element can be accessed and modified through the DOM API, such as adding content, attributes, and styles to the element. It can also be added to the document by using methods such as appendChild() or insertAdjacentHTML().

In the below example we will be creating a dynamic list, all the elements are created using JavaScript, adding the button for interaction when the user wants to add new people to the list.



Learn JavaScript Course

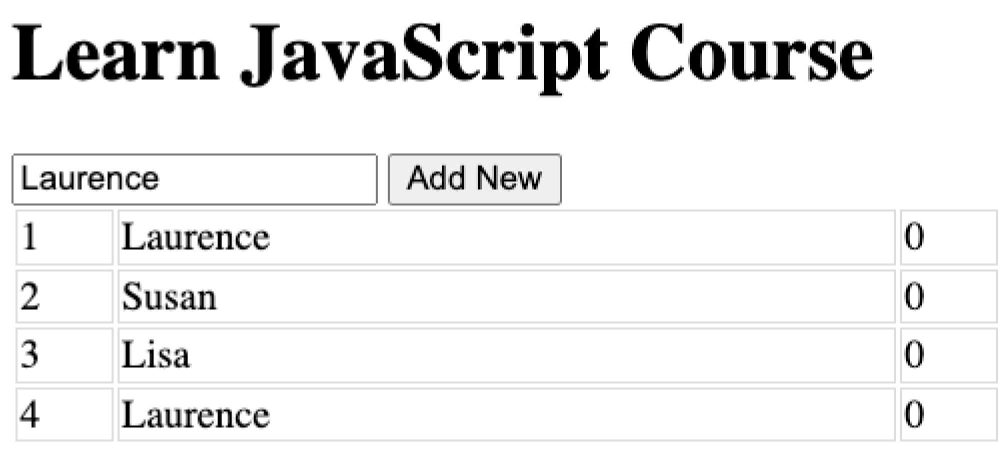
Mike

- Laurence
- Susan
- Lisa
- Lawrence
- Mike

```
const myArr = ['Laurence','Susan','Lisa'];
const output = document.querySelector('.output');

const btn = document.createElement('button');
btn.textContent = 'Add Person';
output.append(btn);

const myInput = document.createElement('input');
myInput.setAttribute('type','text');
myInput.value = 'Lawrence';
output.prepend(myInput);

const ul = document.createElement('ul');
output.append(ul);
build();

btn.addEventListener('click',addPerson);

function addPerson(){
   const newPerson = myInput.value;
   myArr.push(newPerson);
   adder(newPerson);
   console.log(myArr);
}

function adder(person){
   const li = document.createElement('li');
   li.textContent = person;
   ul.append(li);
}

function build(){
   myArr.forEach(ele => {
       adder(ele);
   })
}
```

Create an interactive table list of item object values from a JavaScript array.

https://youtu.be/4Pvz_ILMEdE


Learn JavaScript Course

Laurence

1	Laurence	0
2	Susan	0
3	Lisa	0
4	Laurence	0

Create a list of items within a table using JavaScript. Data is contained within an array with object values.

```
<!DOCTYPE html>
<html>
<head>
  <title>Learn JavaScript</title>
  <style>
    table{
      width:100%;
    }
    td:first-child{
      width:10%;
    }
    td:last-child{
      width:10%;
    }
    td{
      border: 1px solid #ddd;
    }
  </style>
</head>
```

```
<body>
  <h1>Learn JavaScript Course </h1>
  <div>
     <input type="text" id="addFriend" >
     <input type="button" id="addNew" value="Add New">
     <div class="output"></div>
  </div>
  <script src="app10.js"></script>
</body>
</html>

const myArr = [
  {name:'Laurence',score:0,id:1} ,
  {name:'Susan',score:0,id:2} ,
  {name:'Lisa',score:0,id:3}
];
const output = document.querySelector('.output');
const btn = document.querySelector('#addNew');
const addFriend = document.querySelector('#addFriend');
const tblOutput = document.createElement('table');
output.append(tblOutput);
addFriend.value = 'Laurence';
build();

btn.addEventListener('click',()=>{
  const myObj =  {name:addFriend.value,score:0,id:myArr.length+1} ;
  myArr.push(myObj );
  console.log(myArr);
  build();
})
```

```
function build(){
  tblOutput.innerHTML = '';
  myArr.forEach((ele,ind) =>{
     const tr = document.createElement('tr');
     tblOutput.append(tr);
     const td1 = document.createElement('td');
     td1.textContent = ele.id;
     tr.append(td1);
     const td2 = document.createElement('td');
     td2.textContent = ele.name;
     tr.append(td2);
     const td3 = document.createElement('td');
     td3.textContent = ele.score;
     tr.append(td3);
     tr.addEventListener('click',()=>{
        ele.score++;
        td3.textContent = ele.score;
     })
  })

}
```

How to Create Page Elements with JavaScript

Create Page Elements with JavaScript

https://youtu.be/x8STY2Bat-Y

How to Create Page Elements and make them Interactive with Event LIsteners

There are several ways to create page elements with JavaScript, including:

Using the document.createElement() method, which creates a new element with the specified tag name. For example, the following code creates a new div element:

```
let newDiv = document.createElement("div");
```

Using the innerHTML property to add HTML content to an existing element. For example, the following code adds a new p element to an existing div element with an id of "container":

```
let container = document.getElementById("container");
container.innerHTML += "<p>Hello World</p>";
```

Using the appendChild() method to add a new element as a child of an existing element. For example, the following code adds a new p element as a child of an existing div element with an id of "container":

```
let container = document.getElementById("container");
let newP = document.createElement("p");
newP.innerHTML = "Hello World";
container.appendChild(newP);
```

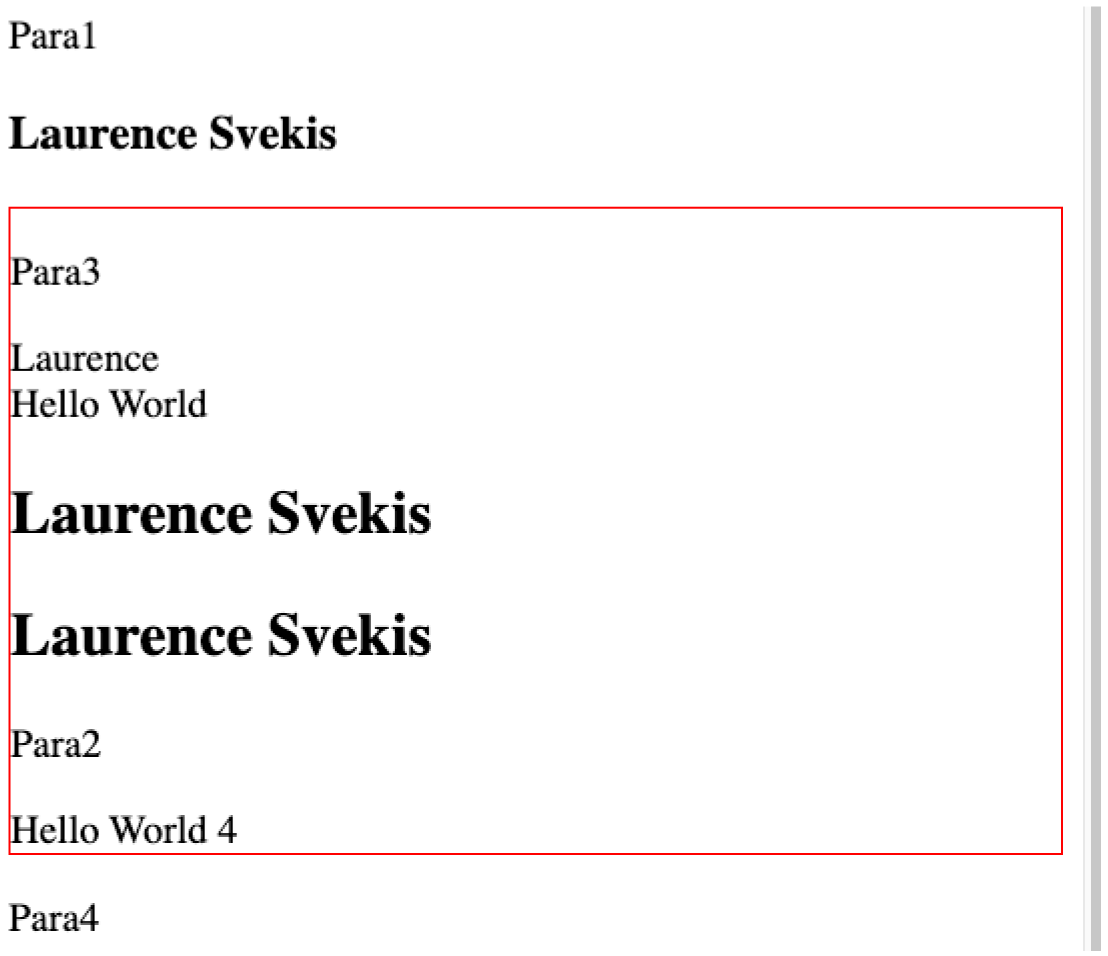
Using the insertAdjacentHTML() method to insert HTML content at a specific position relative to an existing element. For example, the following code adds a new p element before an existing div element with an id of "container":

```
let container = document.getElementById("container");
container.insertAdjacentHTML("beforebegin", "<p>Hello
World</p>");
```

You can also use any of the above methods to add CSS styles, classes and attributes to the newly created elements.

Coding Example of how to insert page content , html elements into your DOM page.

Coding Exercise to demo how to insert HTML page elements into the page with JavaScript


```
Para1

Laurence Svekis

Para3

Laurence
Hello World

Laurence Svekis

Laurence Svekis

Para2

Hello World 4

Para4
```

```
const ele1 = document.createElement('div');
ele1.textContent = 'My new element';
document.body.prepend(ele1);

const output = document.querySelector('.output');
output.innerHTML += '<div>Laurence</div>';
output.innerHTML += '<div>Hello World</div>';
output.style.border = '1px solid red';

const ele2 = document.createElement('h2');
ele2.innerHTML = 'Laurence Svekis';
const el = output.appendChild(ele2);
console.log(el);
```

```
const ele3 = document.createElement('h2');
ele3.innerHTML = 'Laurence Svekis';
const el2 = output.append(ele3);
console.log(el2);

output.insertAdjacentHTML('beforebegin','<p>Para1</p>');
output.insertAdjacentHTML('beforeend','<p>Para2</p>');
output.insertAdjacentHTML('afterbegin','<p>Para3</p>');
output.insertAdjacentHTML('afterend','<p>Para4</p>');

const ele4 = document.createElement('h3');
ele4.textContent = 'Laurence Svekis';
output.insertAdjacentElement('beforebegin',ele4);
output.insertAdjacentText('beforeend','Hello World 4');
```

The JavaScript Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the structure of a document as a tree of objects, with each object (or “node”) representing a part of the document, such as an element, an attribute, or text content. The DOM allows developers to access and manipulate the contents of a document, as well as respond to events and create dynamic content.

The DOM is typically used in combination with JavaScript, as it allows developers to programmatically change the structure and content of a document. For example, a developer might use the DOM to change the text of a heading, add a new element to the page, or change the style of an element.

To work with the DOM, a developer must first access the root node of the document, which is typically the `<html>` element. From there, they can traverse the tree of nodes to access and manipulate specific elements of the document. The DOM provides a number of methods and properties that can be used to access and manipulate the nodes of the tree.

The DOM also provides a way to respond to events that occur in the document, such as a user clicking on a button or a page finishing loading. Developers can use event listeners to listen for specific events and then execute code in response.

In summary, the JavaScript DOM is a programming interface for HTML and XML documents that allows developers to access, manipulate, and respond to the contents of a document. It represents the structure of a document as a tree of objects, with each object representing a part of the document, it allows developers to programmatically change the structure and content of a document, and respond to events on the page.

examples of using the DOM with JavaScript:

1. Accessing an element by its ID:

Copy code
```
// Get the element with the ID "myHeading"
var heading = document.getElementById("myHeading");

// Change the text of the element
heading.innerHTML = "New heading text";
```

2. Accessing elements by their class name:

Copy code
```
// Get all elements with the class "myClass"
```

```
var elements = document.getElementsByClassName("myClass");

// Change the text of the first element with the class "myClass"
elements[0].innerHTML = "New text";
```

3. Adding an event listener:

Copy code
```
// Get the button element
var button = document.getElementById("myButton");

// Add a click event listener to the button
button.addEventListener("click", function() {
  alert("Button was clicked!");
});
```

4. Changing the style of an element:

Copy code
```
// Get the element
var element = document.getElementById("myElement");

// Change the background color of the element
element.style.backgroundColor = "blue";
```

5. Traversing the DOM tree

Copy code
```
// Get the first child of an element
var firstChild = element.firstChild;

// Get the parent of an element
var parent = element.parentNode;

// Get the next sibling of an element
var nextSibling = element.nextSibling;
```

Please note that the above code is just an example and it will only work if there's an existing html structure that matches the selectors. Also keep in mind that the examples are simplified and in real-world scenario's you'll probably need to add error handling and conditionals to make sure you're selecting the right elements.

6. Creating a new element:

Copy code
```
// Create a new <p> element
var newParagraph = document.createElement("p");

// Add text to the new element
newParagraph.innerHTML = "This is a new paragraph";

// Append the new element to the body
document.body.appendChild(newParagraph);
```

7. Removing an element:

Copy code
```
// Get the element to remove
var elementToRemove = document.getElementById("elementToRemove");

// Remove the element
elementToRemove.parentNode.removeChild(elementToRemove);
```

8. Replacing an element:

Copy code
```
// Get the element to replace
var elementToReplace = document.getElementById("elementToReplace");

// Create a new element
```

```
var newElement = document.createElement("div");

// Replace the old element with the new element
elementToReplace.parentNode.replaceChild(newElement, elementToReplace);
```

9. Accessing the value of an input element:

Copy code
```
// Get the input element
var input = document.getElementById("myInput");

// Get the value of the input
var inputValue = input.value;

// Set the value of the input
input.value = "New value";
```

10. Changing the class of an element:

Copy code
```
// Get the element
var element = document.getElementById("myElement");

// Add a class to the element
element.classList.add("new-class");

// Remove a class from the element
element.classList.remove("old-class");

// Toggle a class on an element
element.classList.toggle("toggle-class");
```

11. Changing the attribute of an element:

Copy code
```
// Get the element
```

```
var element = document.getElementById("myElement");

// Get the value of an attribute
var attributeValue = element.getAttribute("href");

// Set the value of an attribute
element.setAttribute("href", "https://www.example.com");
```

12. Getting the position of an element:

Copy code
```
// Get the element
var element = document.getElementById("myElement");

// Get the position of the element
var rect = element.getBoundingClientRect();

// Get the x and y position
var xPos = rect.left + window.pageXOffset;
var yPos = rect.top + window.pageYOffset;
```

13. Changing the CSS of an element:

Copy code
```
// Get the element
var element = document.getElementById("myElement");

// Change the CSS of the element
element.style.cssText = "color: red; font-size: 20px;";
```

14. Traversing the DOM tree using `querySelector`:

Copy code
```
// Get the first element that matches the selector
var element = document.querySelector("#myId .myClass");
```

```
// Get all the elements that match the selector
var elements = document.querySelectorAll(".myClass");
```

15. DOM to manipulate a table:

Copy code
```
// Get the table element
var table = document.getElementById("myTable");

// Get the first row of the table
var firstRow = table.rows[0];

// Get the first cell of the first row
var firstCell = firstRow.cells[0];

// Change the text of the first cell
firstCell.innerHTML = "New text";

// Add a new row to the table
var newRow = table.insertRow(-1);

// Add a new cell to the new row
var newCell = newRow.insertCell(0);

// Add text to the new cell
newCell.innerHTML = "New cell text";
```

Please note that the above code is just an example and it will only work if there’s an existing html structure that matches the selectors. Also keep in mind that the examples are simplified and in real-world scenario’s you’ll probably need to add error handling and conditionals to make sure you’re selecting the right elements. Also, note that the `table.rows` property returns an HTMLCollection of rows in the table, while the `table.insertRow()` method allows you to insert a new row at a specific position in the table.

More Examples of DOM and Table updates

Get all the rows in a table:

```
Copy code// Get the table
var table = document.getElementById("myTable");

// Get all the rows in the table
var rows = table.getElementsByTagName("tr");
```

Delete a row from a table:

```
Copy code// Get the row to delete
var rowToDelete = table.rows[3];

// Delete the row
table.deleteRow(rowToDelete.rowIndex);
```

Modifying table cells:

```
Copy code// Get the cell
var cell = table.rows[2].cells[1];

// Change the text of the cell
cell.innerHTML = "New cell text";

// Change the background color of the cell
cell.style.backgroundColor = "lightgray";
```

Get all the cells in a row:

```
Copy code// Get the row
var row = table.rows[2];

// Get all the cells in the row
var cells = row.getElementsByTagName("td");
```

Get all the cells in a column:

```
Copy code// Get the cells of the first column
var columnCells = table.getElementsByTagName("th");

for(var i = 0; i < columnCells.length; i++){
  console.log(columnCells[i].innerHTML);
}
```

Modifying table headers:

```
Copy code// Get the header cell
var header = table.rows[0].cells[1];

// Change the text of the header
header.innerHTML = "New header text";

// Change the font size of the header
header.style.fontSize = "20px";
```

Add a new table column:

```
Copy code// Add a new table column
var newCol =
table.createTHead().insertRow(-1).insertCell(-1);

// Add text to the new column
newCol.innerHTML = "New Column";
```

Delete a table column:

```
Copy code// Get the column to delete
var colToDelete = table.rows[0].cells[1];

// Delete the column
colToDelete.parentNode.removeChild(colToDelete);
```

DOM examples

Accessing the text content of an element:

```
Copy code// Get the element
var element = document.getElementById("myElement");

// Get the text content of the element
var textContent = element.textContent;

// Set the text content of the element
element.textContent = "New text content";
```

Changing the value of a select element:

```
Copy code// Get the select element
var select = document.getElementById("mySelect");

// Get the selected option
var selectedOption = select.options[select.selectedIndex];

// Change the selected option
select.selectedIndex = 2;
```

Checking if an element has a specific class:

```
Copy code// Get the element
var element = document.getElementById("myElement");

// Check if the element has the class "myClass"
if (element.classList.contains("myClass")) {
  console.log("The element has the class 'myClass'.");
}
```

Getting the children of an element:

```
Copy code// Get the parent element
```

```
var parent = document.getElementById("myParent");

// Get the children of the parent element
var children = parent.children;
```

Cloning an element:

```
Copy code// Get the element to clone
var element = document.getElementById("myElement");

// Clone the element
var clonedElement = element.cloneNode(true);

// Append the cloned element to the body
document.body.appendChild(clonedElement);
```

Getting the previous and next siblings of an element:

```
Copy code// Get the element
var element = document.getElementById("myElement");

// Get the previous sibling of the element
var previousSibling = element.previousElementSibling;

// Get the next sibling of the element
var nextSibling = element.nextElementSibling;
```

Getting the parent element of an element:

```
Copy code// Get the element
var element = document.getElementById("myElement");

// Get the parent element
var parent = element.parentElement;
```

Modifying the HTML content of an element:

```
Copy code// Get the element
var element = document.getElementById("myElement");

// Get the HTML content of the element
var innerHTML = element.innerHTML;

// Set the HTML content of the element
element.innerHTML = "<p>New HTML content</p>";
```

Setting a data attribute of an element:

```
Copy code// Get the element
var element = document.getElementById("myElement");

// Set a data attribute of the element
element.setAttribute("data-my-attribute", "some value");

// Get the value of the data attribute
var dataAttributeValue =
element.getAttribute("data-my-attribute");
```

Hiding an element:

```
Copy code// Get the element to hide
var element = document.getElementById("myElement");

// Hide the element
element.style.display = "none";
```

Showing a hidden element:

```
Copy code// Get the hidden element
var hiddenElement =
document.getElementById("myHiddenElement");

// Show the element
```

```
hiddenElement.style.display = "block";
```

Creating a new option in a select element:

```
Copy code// Get the select element
var select = document.getElementById("mySelect");

// Create a new option element
var option = document.createElement("option");

// Set the value and text of the option
option.value = "newOptionValue";
option.text = "New Option";

// Add the option to the select element
select.add(option);
```

Removing an option from a select element:

```
Copy code// Get the select element
var select = document.getElementById("mySelect");

// Get the option to remove
var optionToRemove = select.options[1];

// Remove the option from the select element
select.remove(optionToRemove.index);
```

Getting the selected option from a select element:

```
Copy code// Get the select element
var select = document.getElementById("mySelect");

// Get the selected option
var selectedOption = select.options[select.selectedIndex];
```

```
// Get the value and text of the selected option
var selectedValue = selectedOption.value;
var selectedText = selectedOption.text;
```

Creating a new form element:

```
Copy code// Create a new input element
var input = document.createElement("input");

// Set the type and value of the input
input.type = "text";
input.value = "New Input";

// Append the input to the form
var form = document.getElementById("myForm");
form.appendChild(input);
```

Getting the value of a form element:

```
Copy code// Get the form element
var element = document.getElementById("myFormElement");

// Get the value of the form element
var value = element.value;
```

Setting the value of a form element:

```
Copy code// Get the form element
var element = document.getElementById("myFormElement");

// Set the value of the form element
element.value = "New Value";
```

Creating a new image element:

```
Copy code// Create a new image element
var img = document.createElement("img");
```

```
// Set the source and alt text of the image
img.src = "image.jpg";
img.alt = "New Image";

// Append the image to the body
document.body.appendChild(img);
```

Using querySelectorAll to get all elements with a specific class:

```
Copy code// Get all elements with class "my-class"
var elements = document.querySelectorAll(".my-class");

// Change the text of all elements
for (var i = 0; i < elements.length; i++) {
  elements[i].innerHTML = "New text";
}
```

Using innerHTML to insert HTML content:

```
Copy code// Get the element
var element = document.getElementById("myElement");

// Insert HTML content
element.innerHTML = "<p>This is a new paragraph.</p><img
src='image.jpg'>";
```

Using textContent to insert text content:

```
Copy code// Get the element
var element = document.getElementById("myElement");

// Insert text content
element.textContent = "This is some text.";
```

Using createTextNode to create a text node:

```
Copy code// Create a text node
var textNode = document.createTextNode("This is some
text.");

// Append the text node to an element
var element = document.getElementById("myElement");
element.appendChild(textNode);
```

Using getElementsByTagName to get elements by tag name:

```
Copy code// Get all <p> elements
var elements = document.getElementsByTagName("p");

// Change the text of all <p> elements
for (var i = 0; i < elements.length; i++) {
  elements[i].innerHTML = "New text";
}
```

Using getElementsByName to get elements by name:

```
Copy code// Get all elements with name "myName"
var elements = document.getElementsByName("myName");

// Change the value of all elements
for (var i = 0; i < elements.length; i++) {
  elements[i].value = "New value";
}
```

Using parentNode to access the parent of an element:

```
Copy code// Get the element
var element = document.getElementById("myElement");
```

```
// Get the parent of the element
var parent = element.parentNode;

// Change the class of the parent
parent.classList.add("new-class");
```

Using children to access the children of an element:

```
Copy code// Get the element
var element = document.getElementById("myElement");

// Get the children of the element
var children = element.children;

// Change the text of the first child
children[0].innerHTML = "New text";
```

Using nextSibling to access the next sibling of an element:

```
Copy code// Get the element
var element = document.getElementById("myElement");

// Get the next sibling of the element
var nextSibling = element.nextSibling;

// Change the text of the next sibling
nextSibling.innerHTML = "New text";
```