JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

```
function myFunction(p1, p2) {
  return p1 * p2;   // The function returns the product of p1 and p2
}
```

JavaScript Function Syntax:

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

A JavaScript function is defined with the **function keyword**, followed by a **name**, followed by **parentheses ()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...)

The code to be executed, by the function, is placed inside curly brackets: {}

Function parameters are listed inside the parentheses () in the function definition.

Function arguments are the values received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

Why do we need functions in JS ?

You can reuse code: Define the code once, and use it many times.
You can use the same code many times with different arguments, to produce different results.

Let’s see by an example:
Suppose we have an array, we want to duplicate the array to create a new array, each element of the new array will be multiplied by 2.

```
// Copy an array and multiply by 2
let arr01 = [1,2,3,4,5];
let arr01dup = [];
for(let i=0;i<arr01.length;i++){
    arr01dup[i] = arr01[i] * 2;
}

console.log(arr01);
console.log(arr01dup);


// Copy another array and multiply by 2
let arr02 = [10,20,30,40,50];
let arr02dup = [];
for(let i=0;i<arr02.length;i++){
    arr02dup[i] = arr02[i] * 2;
}

// In the above code, we're breaking a principle - DRY(Don't repeat yourself)
// Instead we can write a function that works for any array.
```

```
function copyArrayAndMultiplyby2(array){
    let arraydup = [];
    for(let i=0;i<array.length;i++){
        arraydup[i] = array[i] * 2;
    }
    return arraydup;
}

let arr03 = [1,2,3,4,5];
console.log(copyArrayAndMultiplyby2(arr03));
let arr04 = [6,7,8,9,10];
console.log(copyArrayAndMultiplyby2(arr04));
```

Function Return:

When JavaScript reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a return value. The return value is "returned" back to the "caller".

Return value can also be an expression.

Returning a value	Returning an expression

function sum(x,y){
        let z = x+y;
        return z;
}
	function sum(x,y){
	        return (x+y);
	}

Function Invocation

The code inside the function will execute when we invoke (call) the function.

When an event occurs (when a user clicks a button)
When it is invoked (called) from JavaScript code
Automatically (self invoked)

The () Operator Invokes the Function.

N.B: Accessing a function without () will return the function definition instead of the function result.

Example of clicking a Button and calling a function on click:

HTML:

```
<button onclick="myFunction2(10,3)">Try it 2</button>
<div id="id2" class="cl2"></div>
```

JS:

```
function myFunction2(num1,num2){
        let addedval = num1+num2;
        document.getElementById("id2").innerHTML = addedval;
}
```

Variable Scope:

To work with JavaScript efficiently, one of the first things you need to understand is the concept of variable scope. The scope of a variable is controlled by the location of the variable declaration, and defines the part of the program where a particular variable is accessible.

Scoping rules vary from language to language. JavaScript has two scopes – global and local. Any variable declared outside of a function belongs to the global scope, and is therefore accessible from anywhere in your code. Each function has its own scope, and any variable declared within that function is only accessible from that function and any nested functions. Because local scope in JavaScript is created by functions, it’s also called function scope.

Functions can also be assigned to Variables.

```
function sum(x,y){
        return (x+y);
}

// We can call the function and assign the returned value to a variable.
let s1 = sum(5,5);
console.log(s1);
// or we can assign the entire function definition to a variable.
// And we can use that variable as a function call.
let s2 = sum;
console.log(s2(10,20));
```

JavaScript Anonymous functions:

An anonymous function is a function that was declared without any named identifier to refer to it. As such, an anonymous function is usually not accessible after its initial creation.

Normal function definition:

```
function hello() {
  alert('Hello world');
}
hello();
```

Anonymous function definition:

```
var anon = function() {
  console.log('I am anonymous');
}
anon();
```

Another example:

```
var subst = function(num1,num2) {
  return (num1-num2);
}
console.log(subst(3,2));
```

N.B: An Anonymous function must be assigned to a variable. Otherwise we have to give the function a name or make it a Self-Executing Anonymous Function.

Self-Executing Anonymous Function:

A JavaScript function that runs as soon as it is defined. Also known as an IIFE (Immediately Invoked Function Expression).

How do we create Self-Executing Anonymous Function?

We have to wrap the entire function inside a parenthesis.
Then we have to add the auto invoke operator(), to call it immediately after the definition.

Example:

```
(function(){
        console.log("this is self executing anonymous function");
})();
```

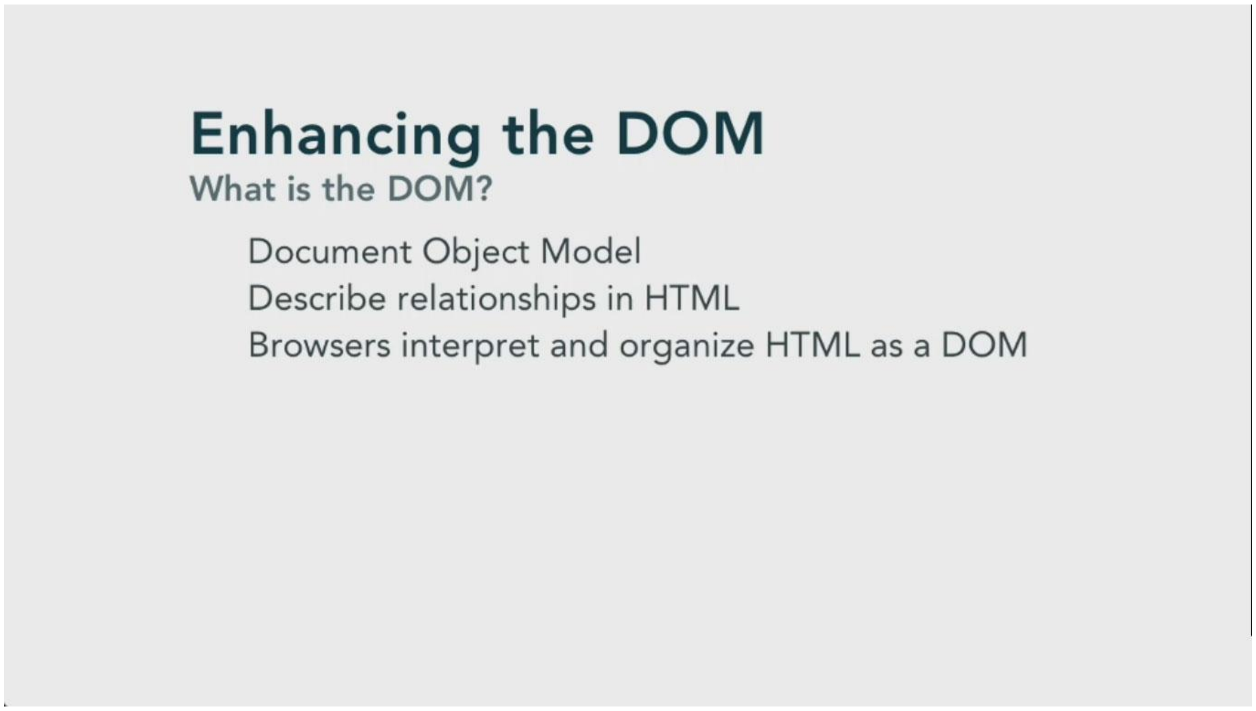
// Self-Executing Anonymous Function with parameters:

```
(function(x,y){
        z = x*y;
        console.log(z);
})(12,3);
```

// Self-Executing Anonymous Function with parameters and return values :

```
let divide1 = (function(x,y){
        z = x/y;
        return z;
})(12,3);

console.log(divide1);
```



Enhancing the DOM

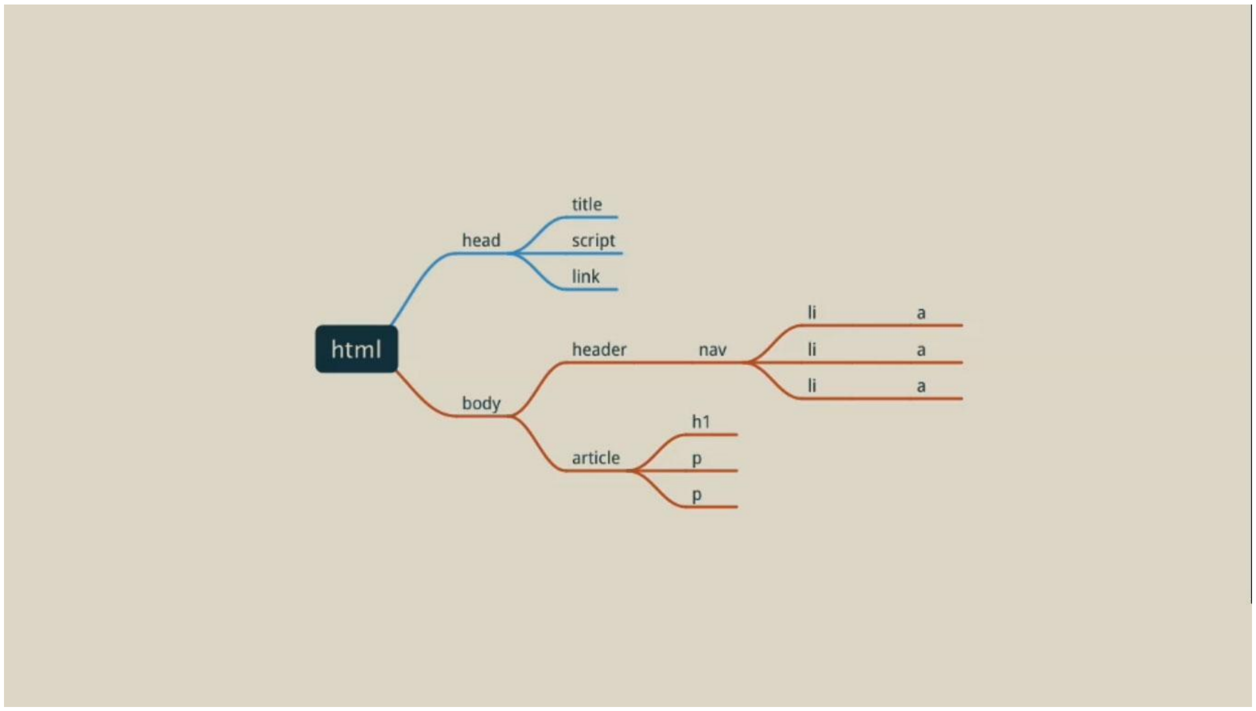
What is the DOM?

Document Object Model

Describe relationships in HTML

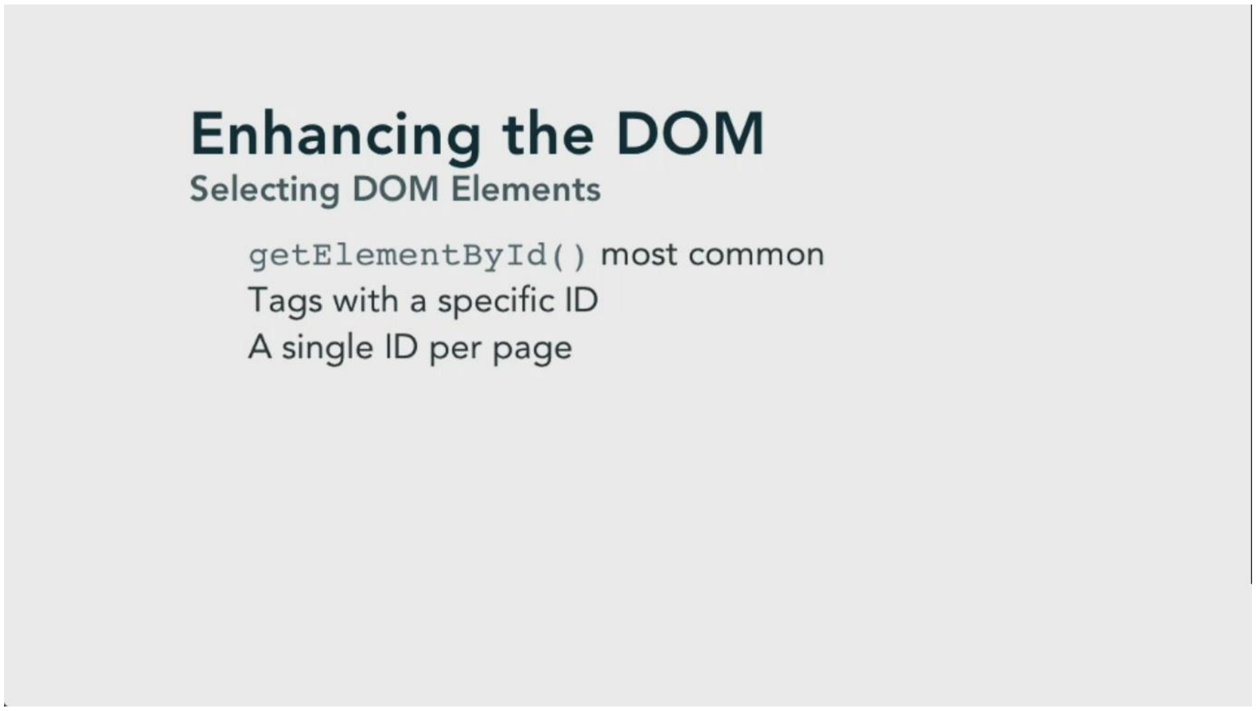
Browsers interpret and organize HTML as a DOM

Every element and texts inside a DOM is called a NODE.



A node can have it’s own child node.

Nodes at the same level are called siblings.



Enhancing the DOM

Selecting DOM Elements

- `getElementById()` most common
- Tags with a specific ID
- A single ID per page

HTML DOM getElementById() Method

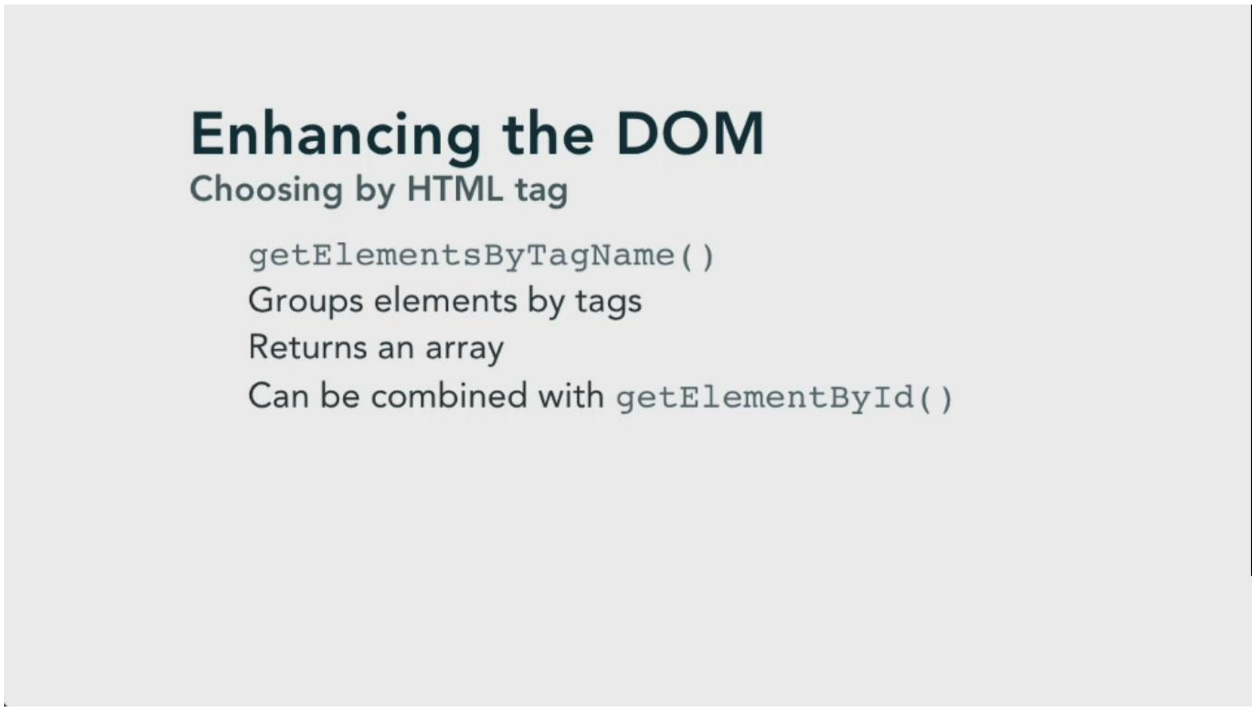
The getElementById() method returns the element that has the ID attribute with the specified value.

This method is one of the most common methods in the HTML DOM, and is used almost every time you want to manipulate, or get info from, an element on your document.

Returns null if no elements with the specified ID exists.

An ID should be unique within a page. However, if more than one element with the specified ID exists, the getElementById() method returns the first element in the source code.

let section01 = document.getElementById('section01');



Enhancing the DOM

Choosing by HTML tag

`getElementsByTagName()`

Groups elements by tags

Returns an array

Can be combined with `getElementById()`

HTML DOM getElementsByTagName() Method:

The getElementsByTagName() method returns a collection of all elements in the document with the specified tag name, as a NodeList object.

The NodeList object represents a collection of nodes. The nodes can be accessed by index numbers. The index starts at 0.

Tip: You can use the length property of the NodeList object to determine the number of elements with the specified tag name, then you can loop through all elements and extract the info you want.

For example:

```
document.getElementsByTagName('section').length
```

// selecting the first section tag from the page and set the font color to red
document.getElementsByTagName('section')[0].style.color = 'red'

// selecting the Second section tag from the page and set the font color to blue
document.getElementsByTagName('section')[1].style.color = 'blue'

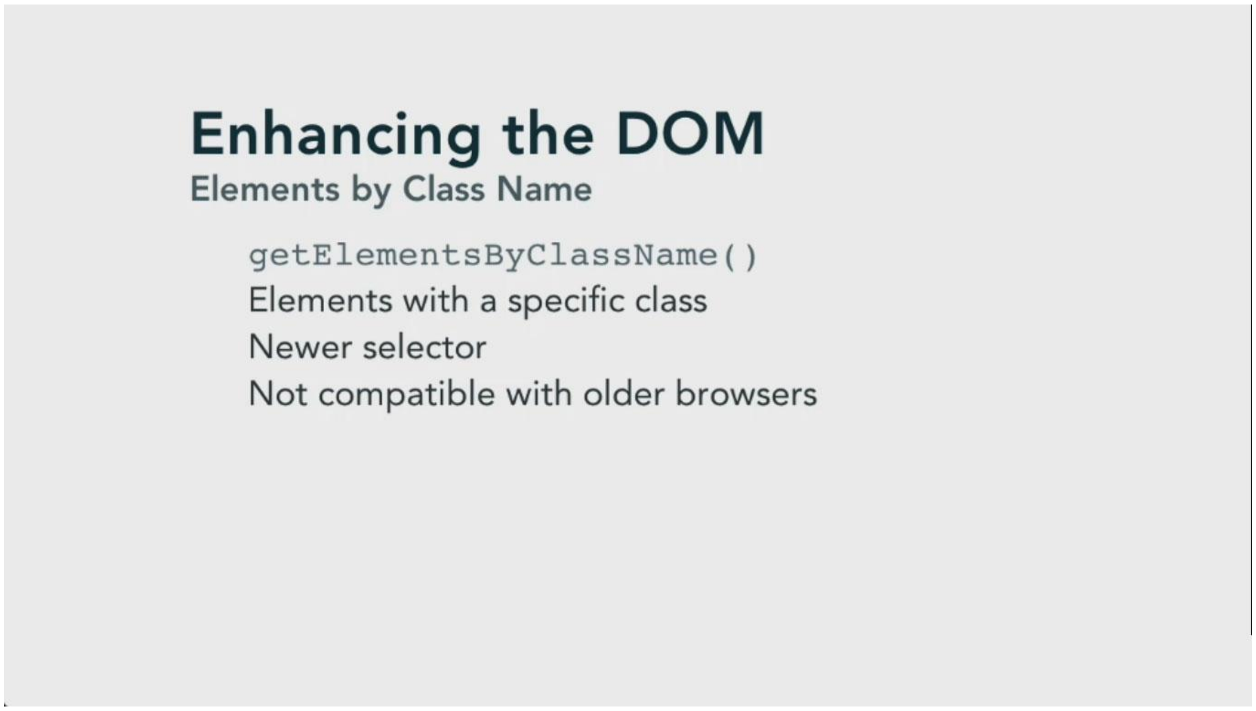
Let’s use document.getElementsByTagName() combined with document.getElementById() :

// this with give the number of all of the li of our document.
console.log(document.getElementsByTagName('li').length);

// this will return the number of the li inside the element with id 'menu_home'.
console.log(document.getElementById('menu_home').getElementsByTagName('li').length);

```
let menuliAll =
document.getElementById('menu_home').getElementsByTagName('li');
menuliAll[0].style.listStyle='none';
menuliAll[0].style.backgroundColor='#ff8800';
menuliAll[0].style.color='#ffffff';
menuliAll[0].style.padding="10px";
menuliAll[0].style.textTransform="uppercase";
menuliAll[0].style.textDecoration="underline";
menuliAll[0].style.textAlign="center";
```

Following statement will give us an error:
menuliAll.style.listStyle='none';



Enhancing the DOM

Elements by Class Name

- `getElementsByClassName()`
- Elements with a specific class
- Newer selector
- Not compatible with older browsers

HTML DOM getElementsByClassName() Method:

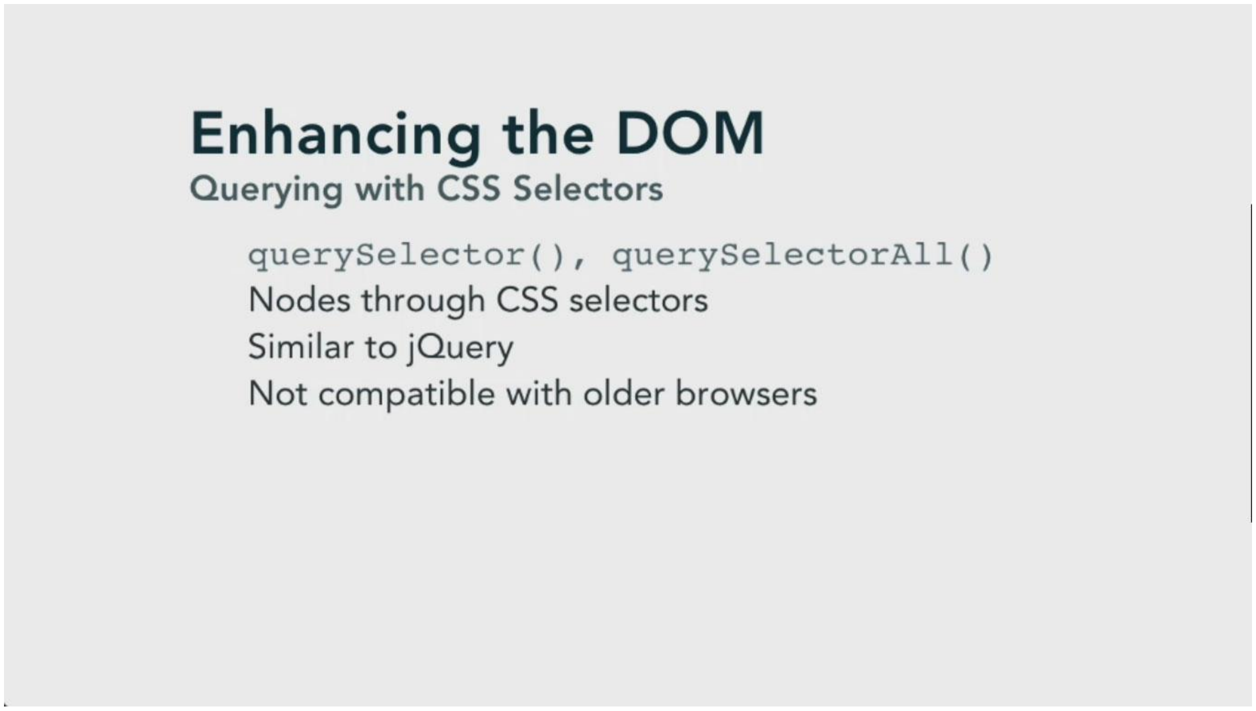
The getElementsByClassName() method returns a collection of all elements in the document with the specified class name, as a NodeList object.

The NodeList object represents a collection of nodes. The nodes can be accessed by index numbers. The index starts at 0.

Tip: You can use the length property of the NodeList object to determine the number of elements with a specified class name, then you can loop through all elements and extract the info you want.

```
// following line will select all of the elements with class - myArticle
let myArticles = document.getElementsByClassName('myArticle');

// we're styling the first element with class - myArticle
myArticles[0].style.backgroundColor = '#dddddd';
```



Enhancing the DOM

Querying with CSS Selectors

`querySelector(), querySelectorAll()`

Nodes through CSS selectors

Similar to jQuery

Not compatible with older browsers

HTML DOM querySelector() Method:

The querySelector() method returns **the first element** that matches a specified CSS selector(s) in the document.

Note: **The querySelector() method only returns the first element that matches the specified selectors.** To return all the matches, use the querySelectorAll() method instead.

```
// select the first article with class - myArticle
let firstArticle = document.querySelector('.myArticle');
// Then changing the background color of that article tag.
firstArticle.style.backgroundColor = 'red';
```

HTML DOM querySelectorAll() Method:

The querySelectorAll() method returns all elements in the document that matches a specified CSS selector(s), as a static NodeList object.

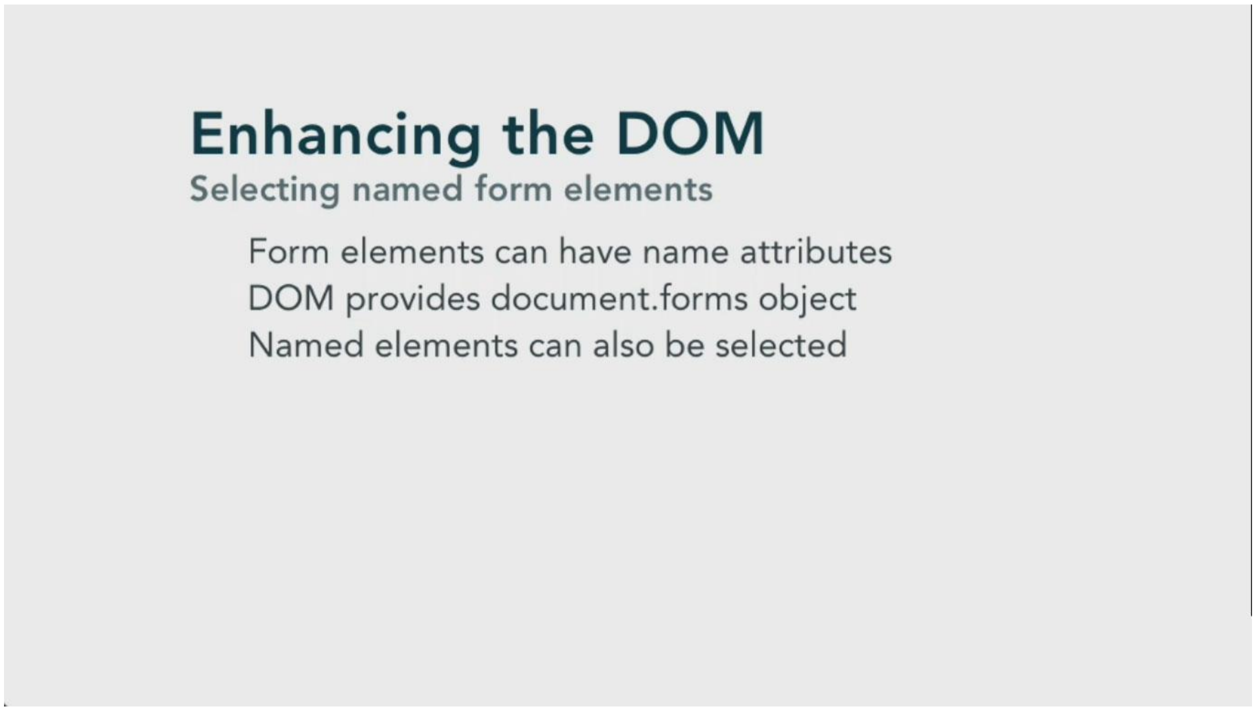
The NodeList object represents a collection of nodes. The nodes can be accessed by index numbers. The index starts at 0.

Code Example:

```
let AllArticles = document.querySelectorAll('.myArticle');

console.log(AllArticles.length);

AllArticles[2].style.backgroundColor = 'gray';
```



Enhancing the DOM

Selecting named form elements

Form elements can have name attributes
DOM provides document.forms object
Named elements can also be selected

document.forms.length === will give us no of forms in our html page.

document.forms[0] = selects the first form of our DOM.
document.forms[1] = selects the second form of our DOM.

We can also access with just the id.

```
// selecting the first form, then a form field by field name and changing the value of that field.
document.forms[0].firstname.value = "Subhasish Nath";
```

```
let radios = document.querySelectorAll("input[type='radio']");
```

```
// console.log(radios[0]);
radios[1].checked = 1;
```

HTML DOM addEventListener() Method:

The addEventListener() method attaches an event handler to the specified element.

```
<!DOCTYPE html>
<html>
<body>

<p>This example uses the addEventListener() method to execute a
function when a user clicks on a button.</p>

<button id="myBtn">Try it</button>

<p id="demo"></p>

<script>
document.getElementById("myBtn").addEventListener("click", myFunction);

function myFunction() {
  document.getElementById("demo").innerHTML = "Hello World";
}
</script>

</body>
</html>
```

You can add events of different types to the same element.

This example demonstrates how to add many events on the same <button> element:

```
document.getElementById("myBtn").addEventListener("mouseover",
myFunction);
document.getElementById("myBtn").addEventListener("click",
someOtherFunction);
document.getElementById("myBtn").addEventListener("mouseout",
someOtherFunction);
```

When passing parameter values, we can use an "anonymous function" that calls the specified function with the parameters:

```
document.getElementById("myBtn").addEventListener("click", function() {
  myFunction(p1, p2);
});
```

Change the background color of a <button> element:

```
document.getElementById("myBtn").addEventListener("click", function(){
  this.style.backgroundColor = "red";
});
```

HTML DOM appendChild() Method:

The appendChild() method appends a node as the last child of a node.

Tip: If you want to create a new paragraph, with text, remember to create the text as a Text node which you append to the paragraph, then append the paragraph to the document.

You can also use this method to move an element from one element to another (See "More Examples").

```
<!DOCTYPE html>
<html>
<body>

<ul id="myList">
  <li>Coffee</li>
  <li>Tea</li>
</ul>

<p>Click the button to append an item to the end of the list.</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  var node = document.createElement("li");
  var textnode = document.createTextNode("Water");
  node.appendChild(textnode);
  document.getElementById("myList").appendChild(node);
}
</script>

<p><strong>Note:</strong><br>First create an LI node,<br> then create a
Text node,<br> then append the Text node to the LI node.<br>Finally
append the LI node to the list.</p>

</body>
</html>
```

HTML DOM insertBefore() Method:

The insertBefore() method inserts a node as a child, right before an existing child, which you specify.

```
<!DOCTYPE html>
<html>
<body>

<ul id="myList">
  <li>Apple</li>
    <li>Lemon</li>
      <li>Coffee</li>
        <li>Orange</li>
  <li>Tea</li>
</ul>

<p>Click the button to insert an item to the list.</p>

<button onclick="myFunction()">Try it</button>

<p><strong>Example explained:</strong><br>First create a LI node,<br>
then create a Text node,<br> then append the Text node to the LI
node.<br>Finally insert the LI node before the first child node in the
list.</p>

<script>
function myFunction() {
  var newItem = document.createElement("LI");
  var textnode = document.createTextNode("Cake");
  newItem.appendChild(textnode);

  var list = document.getElementById("myList");
  list.insertBefore(newItem, list.childNodes[2]);
}
</script>

</body>
</html>
```

HTML DOM removeChild() Method:

The removeChild() method removes a specified child node of the specified element.

Returns the removed node as a Node object, or null if the node does not exist.

Note: The removed child node is no longer part of the DOM. However, with the reference returned by this method, it is possible to insert the removed child to an element at a later time (See "More Examples").

```
<!DOCTYPE html>
<html>
<body>

<ul id="myList"><li>Coffee</li><li>Tea</li><li>Milk</li></ul>

<p>Click the button to remove the first item from the list.</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  var list = document.getElementById("myList");
  list.removeChild(list.childNodes[0]);
}
</script>

</body>
</html>
```

https://codepen.io/cphemm/pen/reNwWd?editors=1000 ==== A fun tip calculater using JS.