5. JavaScript: Advanced

2023학년 2학기 웹응용프로그래밍

권 동 현

Contents

- ES2015+
- Document Object Model (DOM) manipulation
- Events handling
- Form validation

ES2015+

CSLAB

JavaScript History

- JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.
- ECMAScript is the official name of the language.
- ECMAScript versions have been abbreviated to ES1, ES2, ES3, ES5, and ES6.
- Since 2016, versions are named by year (ECMAScript 2016, 2017, 2018, 2019, 2020).

1. let, const

- Before ES6 (2015), JavaScript had **Global Scope** and **Function Scope**.
- ES6 introduced two important new JavaScript keywords: **let** and **const**.
- These two keywords provide **Block Scope** in JavaScript.
  - Variables declared inside a { } block cannot be accessed from outside the block:

- The **let** keyword allows you to declare a variable with block scope.
- The **const** keyword allows you to declare a constant (a JavaScript variable with a constant value).

```
var x = 10;
// Here x is 10
{
  let x = 2;
  // Here x is 2
}
// Here x is 10
```

```
var x = 10;
// Here x is 10
{
  const x = 2;
  // Here x is 2
}
// Here x is 10
```

2. Template Literals

- **Template Literals** use back-ticks (``) rather than the quotes ("") to define a string
  - Template literals provide an easy way to interpolate variables and expressions into strings.
  - The method is called **string interpolation**.

```
let firstName = "John";
let lastName = "Doe";

let text = `Welcome ${firstName}, ${lastName}!`;
```

```
let price = 10;
let VAT = 0.25;

let total = `Total: ${(price * (1 + VAT)).toFixed(2)}`;
```

```
let header = "Templates Literals";
let tags = ["template literals", "javascript", "es6"];

let html = `<h2>${header}</h2><ul>`;
for (const x of tags) {
  html += `<li>${x}</li>`;
}

html += `</ul>`;
```

3. Object Literals

- Expressing object literals with a concise syntax
  - There is no need to add *:function* to an object's method.
    - {sayNode: sayNode} can be abbreviated to { sayNode }
  - Dynamic property names can be used as object property names using [variable+value]

```
var sayNode = function() {
  console.log('Node');
};
var es = 'ES';
var oldObject = {
  sayJS: function() {
    console.log('JS');
  },
  sayNode: sayNode,
};
oldObject[es + 6] = 'Fantastic';
oldObject.sayNode(); // Node
oldObject.sayJS(); // JS
console.log(oldObject.ES6); //
Fantastic
```

```
const newObject = {
  sayJS() {
    console.log('JS');
  },
  sayNode,
  [es + 6]: 'Fantastic',
};
newObject.sayNode(); // Node
newObject.sayJS(); // JS
console.log(newObject.ES6); //
Fantastic
```

4. Arrow Functions

- Arrow functions allows a **short syntax** for writing function expressions.
  - You don't need the **function** keyword, the **return** keyword, and the **curly brackets**.
- Arrow functions do not have their own **this**. They are not well suited for defining object methods.
- Arrow functions are **not hoisted**. They must be defined before they are used.
- Using **const is safer than using var**, because a function expression is always a constant value.
- You **can only omit the return keyword and the curly brackets if the function is a single statement**.
  - Because of this, it might be a good habit to always keep them:

```
// ES5
var mul = function(x, y) {
   return x * y;
}

// ES6
const mul = (x, y) => x * y;

const mul = (x, y) => { return x * y };
```

5. Destructuring assignment

- The **destructuring assignment** syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

```
const obj = { a: 1, b: 2 };
const { a, b } = obj;
// is equivalent to:
// const a = obj.a;
// const b = obj.b;
```

```
const obj = { a: 1, b: { c: 2 } };
const {
  a,
  b: { c: d },
} = obj;
// Two variables are bound: `a` and `d`
```

6. Classes

- JavaScript Classes are templates for JavaScript Objects.
  - Use the keyword class to create a class.
  - Always add a method named constructor()

```javascript
var Human = function(type) {
  this.type = type || 'human';
};

Human.isHuman = function(human) {
  return human instanceof Human;
}

Human.prototype.breathe = function() {
  alert('h-a-a-a-m');
};

var Zero = function(type, firstName, lastName) {
  Human.apply(this, arguments);
  this.firstName = firstName;
  this.lastName = lastName;
};
Zero.prototype = Object.create(Human.prototype);
Zero.prototype.constructor = Zero; // 상속하는 부분
Zero.prototype.sayName = function() {
  alert(this.firstName + ' ' + this.lastName);
};
var oldZero = new Zero('human', 'Zero', 'Cho');
Human.isHuman(oldZero); // true
```

```javascript
class Human {
  constructor(type = 'human') {
    this.type = type;
  }

  static isHuman(human) {
    return human instanceof Human;
  }

  breathe() {
    alert('h-a-a-a-m');
  }
}
class Zero extends Human {
  constructor(type, firstName, lastName) {
    super(type);
    this.firstName = firstName;
    this.lastName = lastName;
  }

  sayName() {
    super.breathe();
    alert(`${this.firstName} ${this.lastName}`);
  }
}

const newZero = new Zero('human', 'Zero', 'Cho');
Human.isHuman(newZero); // true
```

7. Promises

- A Promise is a JavaScript object that links "**Producing Code**" and "**Consuming Code**".
  - *"I Promise a Result!"*
  - "Producing Code" can take some time and "Consuming Code" must wait for the result.

```javascript
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

7. Promises

```
let myPromise = new Promise(function(myResolve, myReject) {
  setTimeout(function() { myResolve("I love You !!"); }, 3000);
});

myPromise.then(function(value) {
  document.getElementById("demo").innerHTML = value;
});
```

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

let myPromise = new Promise(function(myResolve, myReject) {
  let x = 0;

// The producing code (this may take some time)

  if (x == 0) {
    myResolve("OK");
  } else {
    myReject("Error");
  }
});

myPromise.then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
```

8. async/await

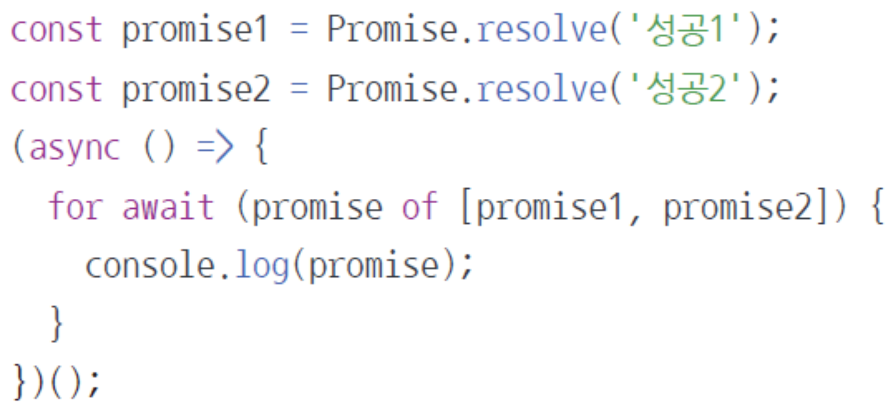
- *"async and await make promises easier to write"*
  - **async** makes a function return a Promise
  - **await** makes a function wait for a Promise

```
async function myFunction() {
  return "Hello";
}
myFunction().then(
  function(value) {myDisplayer(value);}
);
```

```
async function myDisplay() {
  let myPromise = new Promise(function(resolve) {
    setTimeout(function() {resolve("I love You !!");}, 3000);
  });
  document.getElementById("demo").innerHTML = await myPromise;
}

myDisplay();
```

8. async/await

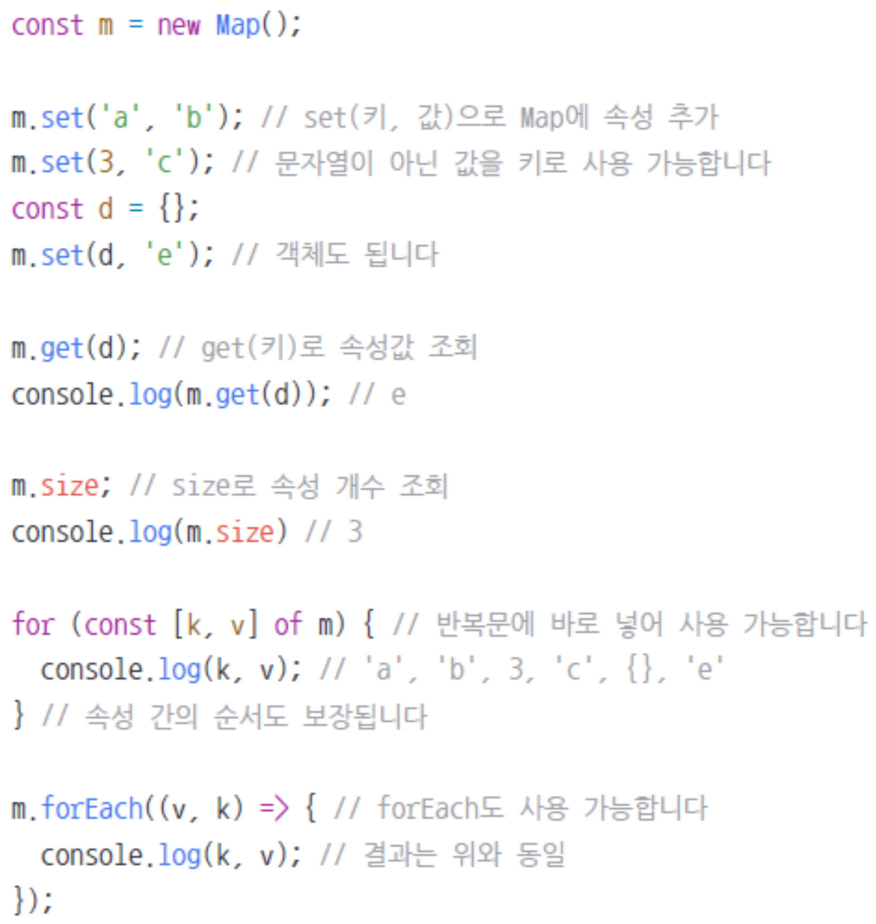
- for await (변수 of 프로미스배열)
  - resolve된 프로미스가 변수에 담겨 나옴
  - await을 사용하기 때문에 async 함수 안에서 해야함



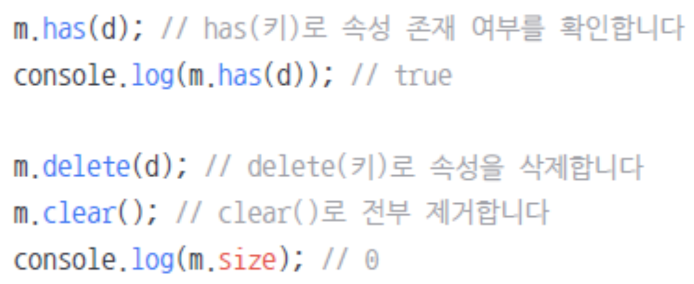
```
const promise1 = Promise.resolve('성공1');
const promise2 = Promise.resolve('성공2');
(async () => {
  for await (promise of [promise1, promise2]) {
    console.log(promise);
  }
})();
```

9. Map/Set

- Map
  - A Map holds key-value pairs where the keys can be any datatype.
  - A Map remembers the original insertion order of the keys.
  - A Map has a property that represents the size of the map.



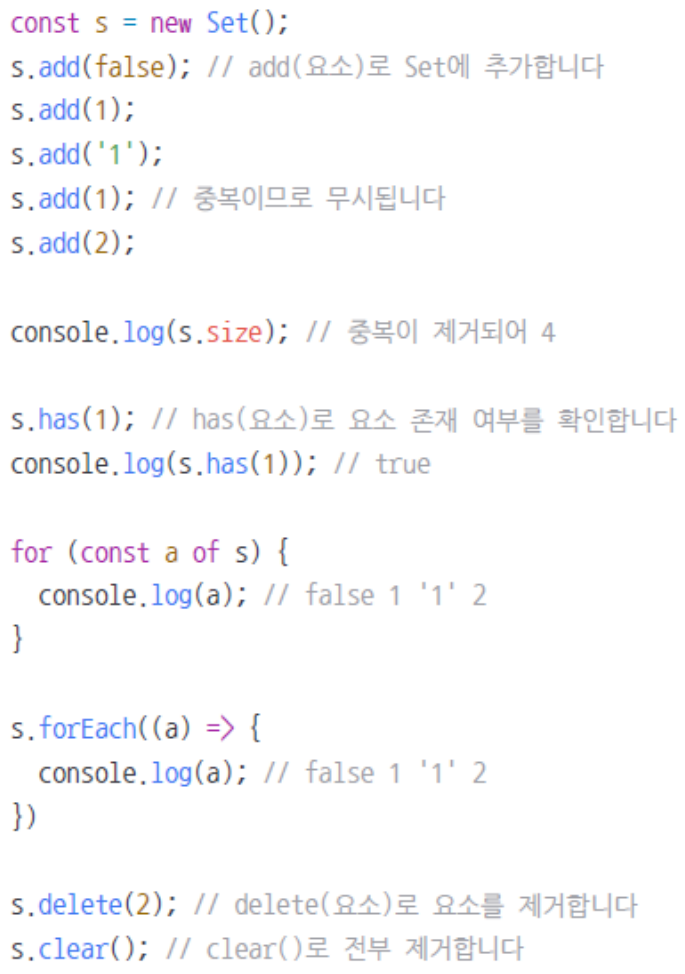
```
const m = new Map();

m.set('a', 'b'); // set(키, 값)으로 Map에 속성 추가
m.set(3, 'c'); // 문자열이 아닌 값을 키로 사용 가능합니다
const d = {};
m.set(d, 'e'); // 객체도 됩니다

m.get(d); // get(키)로 속성값 조회
console.log(m.get(d)); // e

m.size; // size로 속성 개수 조회
console.log(m.size) // 3

for (const [k, v] of m) { // 반복문에 바로 넣어 사용 가능합니다
  console.log(k, v); // 'a', 'b', 3, 'c', {}, 'e'
} // 속성 간의 순서도 보장됩니다

m.forEach((v, k) => { // forEach도 사용 가능합니다
  console.log(k, v); // 결과는 위와 동일
});
```



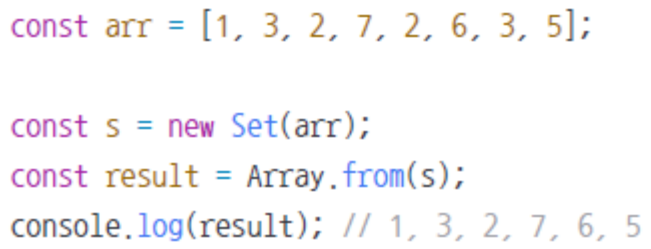
```
m.has(d); // has(키)로 속성 존재 여부를 확인합니다
console.log(m.has(d)); // true

m.delete(d); // delete(키)로 속성을 삭제합니다
m.clear(); // clear()로 전부 제거합니다
console.log(m.size); // 0
```

9. Map/Set

- Set
  - A JavaScript Set is a collection of unique values.
  - Each value can only occur once in a Set.
  - A Set can hold any value of any data type.



```
const s = new Set();
s.add(false); // add(요소)로 Set에 추가합니다
s.add(1);
s.add('1');
s.add(1); // 중복이므로 무시됩니다
s.add(2);

console.log(s.size); // 중복이 제거되어 4

s.has(1); // has(요소)로 요소 존재 여부를 확인합니다
console.log(s.has(1)); // true

for (const a of s) {
  console.log(a); // false 1 '1' 2
}

s.forEach((a) => {
  console.log(a); // false 1 '1' 2
})

s.delete(2); // delete(요소)로 요소를 제거합니다
s.clear(); // clear()로 전부 제거합니다
```



```
const arr = [1, 3, 2, 7, 2, 6, 3, 5];

const s = new Set(arr);
const result = Array.from(s);
console.log(result); // 1, 3, 2, 7, 6, 5
```

10. Nullish Coalescing Operator/ Optional Chaining Operator

- The **Nullish Coalescing Operator(??)** operator returns the first argument if it is not nullish (null or undefined).
- Otherwise it returns the second.

```
let name = null;
let text = "missing";
let result = name ?? text;
```

- The **Optional Chaining Operator(?.)** returns undefined if an object is undefined or null (instead of throwing an error).

```
const car = {type:"Fiat", model:"500", color:"white"};
let name = car?.name;
```

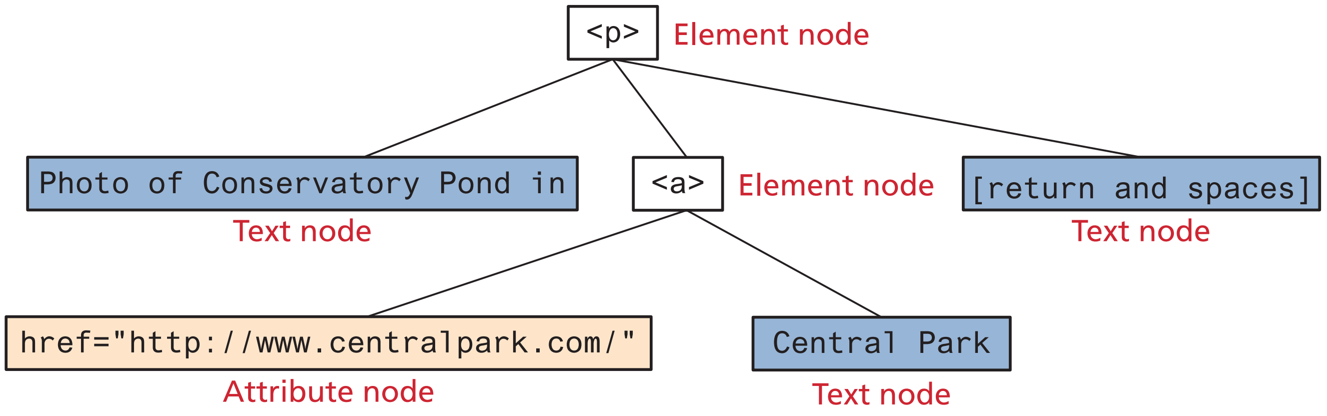
Document Object Model Manipulation

CSLAB

The HTML DOM

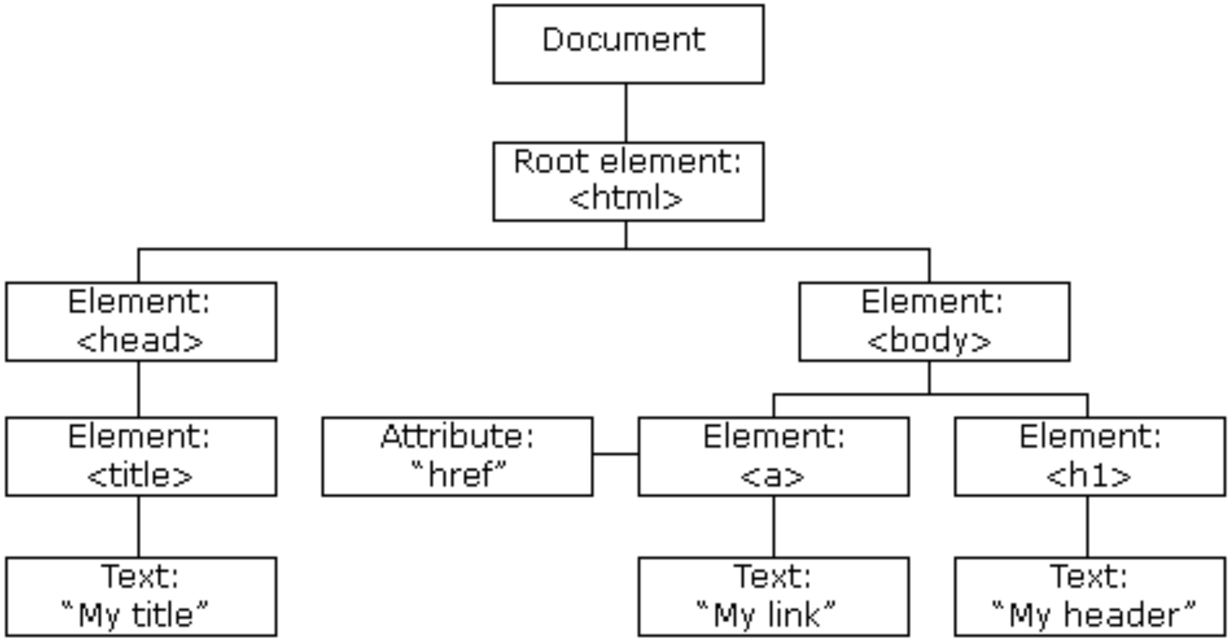
- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- The **HTML DOM** model is constructed as a tree of **Objects**:

```
<p>Photo of Conservatory Pond in
   <a href="http://www.centralpark.com/">Central Park</a>
</p>
```



The HTML DOM

- everything in an HTML document is a node:
  - The entire document is a **document node**
  - Every HTML element is an **element node**
  - The text inside HTML elements are **text nodes**
  - Every HTML attribute is an **attribute node**
  - All comments are **comment nodes**



What is the HTML DOM?

- The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:
  - The HTML elements as **objects.**
  - The **properties** of all HTML elements
  - The **methods** to access all HTML elements
  - The **events** for all HTML elements
- In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

What is the HTML DOM?

- HTML DOM **methods** are **actions** you can perform (on HTML Elements).
- HTML DOM **properties** are **values** (of HTML Elements) that you can set or change.
    - In the example above, *getElementById* is a method, while *innerHTML* is a property.

```
<html>
<body>

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

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

Finding HTML elements

- By ID

```
const element = document.getElementById("intro");
```

- By Tag Name

```
const element = document.getElementsByTagName("p");
```

- By Class Name

```
const x = document.getElementsByClassName("intro");
```

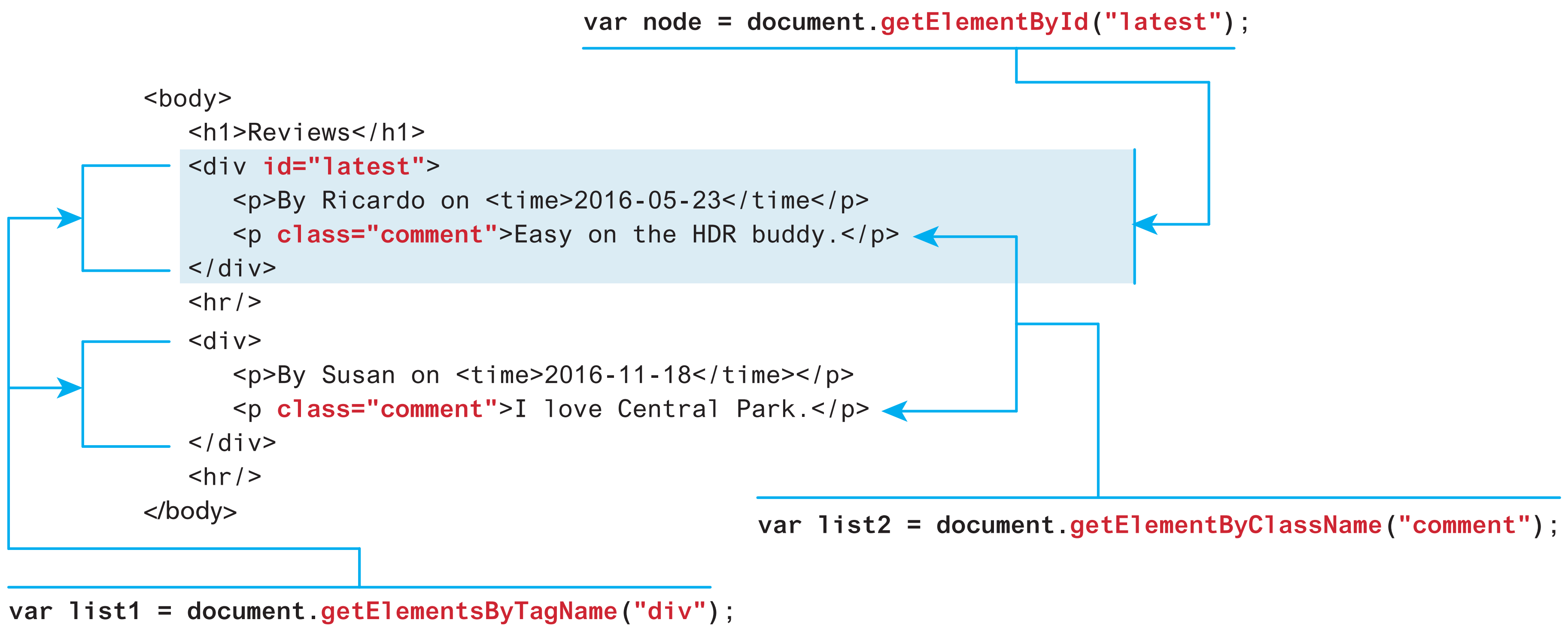
- By CSS Selectors

```
const x = document.querySelectorAll("p.intro");
```

- Example

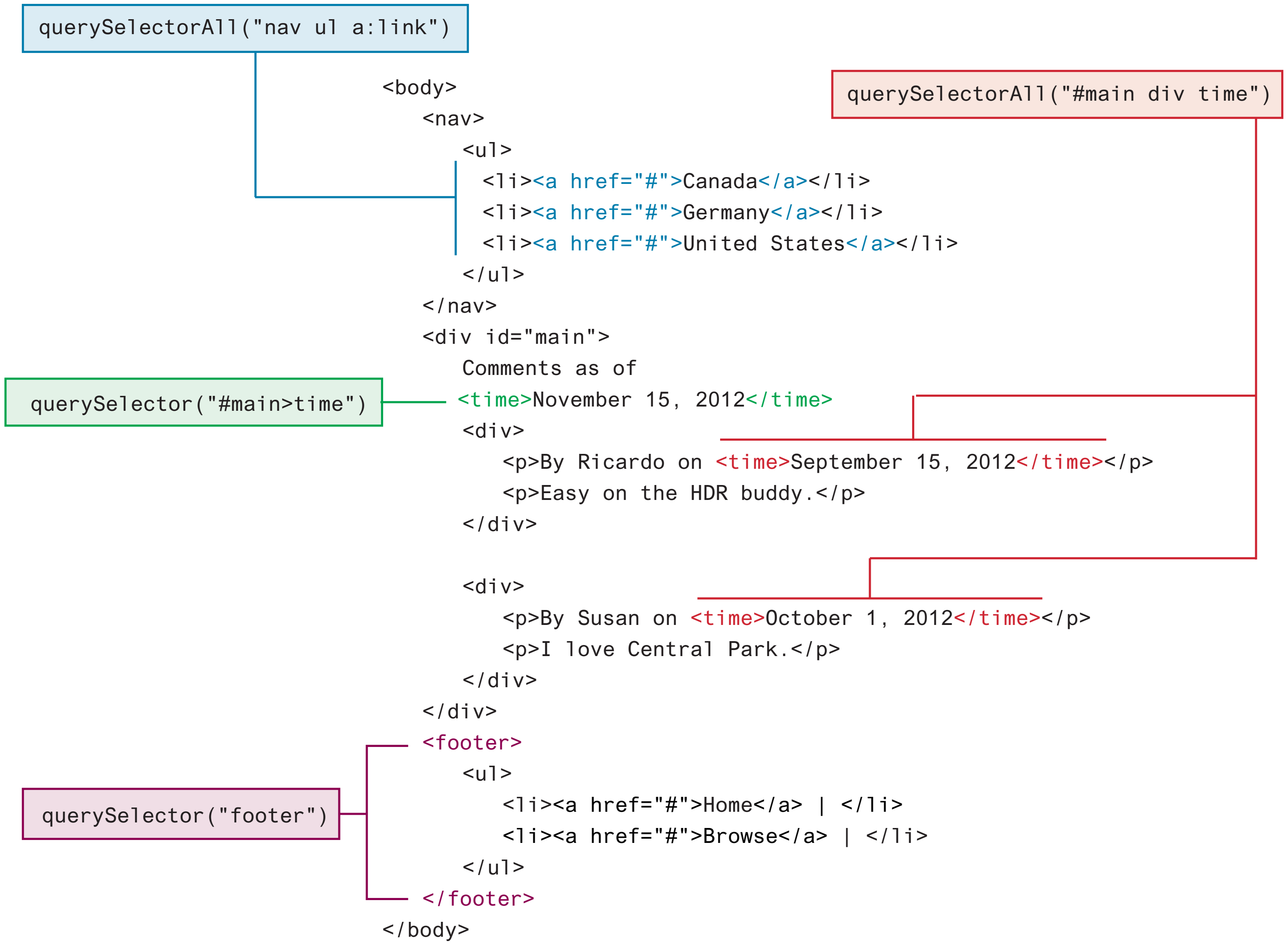
```
const x = document.getElementById("main");
const y = x.getElementsByTagName("p");
```

Finding HTML elements



```
var node = document.getElementById("latest");

<body>
   <h1>Reviews</h1>
   <div id="latest">
      <p>By Ricardo on <time>2016-05-23</time</p>
      <p class="comment">Easy on the HDR buddy.</p>
   </div>
   <hr/>
   <div>
      <p>By Susan on <time>2016-11-18</time></p>
      <p class="comment">I love Central Park.</p>
   </div>
   <hr/>
</body>

var list2 = document.getElementByClassName("comment");

var list1 = document.getElementsByTagName("div");
```

Finding HTML elements


querySelectorAll("nav ul a:link")
querySelectorAll("#main div time")
querySelector("#main>time")
querySelector("footer")

```
<body>
   <nav>
      <ul>
        <li><a href="#">Canada</a></li>
        <li><a href="#">Germany</a></li>
        <li><a href="#">United States</a></li>
      </ul>
   </nav>
   <div id="main">
      Comments as of
      <time>November 15, 2012</time>
      <div>
         <p>By Ricardo on <time>September 15, 2012</time></p>
         <p>Easy on the HDR buddy.</p>
      </div>

      <div>
         <p>By Susan on <time>October 1, 2012</time></p>
         <p>I love Central Park.</p>
      </div>
   </div>
   <footer>
      <ul>
         <li><a href="#">Home</a> | </li>
         <li><a href="#">Browse</a> | </li>
      </ul>
   </footer>
</body>
```

Changing HTML

- Changing HTML Content

```
const element = document.getElementById("id01");
element.innerHTML = "New Heading";

// or
const element = document.getElementById("id01").innerHTML = "New Heading";
```

- Changing the Value of an Attribute

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

<img id="myImage" src="smiley.gif">

<script>
document.getElementById("myImage").src = "landscape.jpg";
</script>

</body>
</html>
```

Element node Object

Property	Description
className	The current value for the class attribute of this HTML element.
id	The current value for the id of this element.
innerHTML	Represents all the things inside of the tags. This can be read or written to and is the primary way which we update particular div's using JS.
style	The style attribute of an element. We can read and modify this property.
tagName	The tag name for the element.

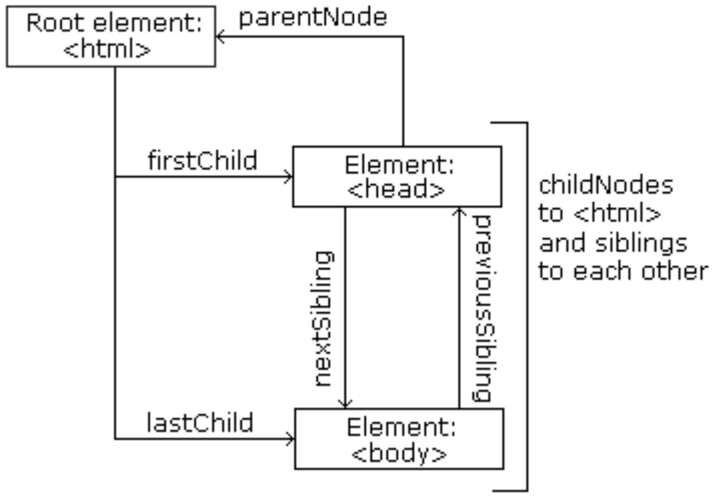
More Properties

Property	Description	Tags
href	The href attribute used in a tags to specify a URL to link to.	a
name	The name property is a bookmark to identify this tag. Unlike id which is available to all tags, name is limited to certain form related tags.	a, input, textarea, form
src	Links to an external URL that should be loaded into the page (as opposed to href which is a link to follow when clicked)	img, input, iframe, script
value	The value is related tot he value attribute of input tags. Often the value of an input field is user defined, and we use value to get that user input.	Input, textarea, submit

DOM navigation

- With the HTML DOM, you can navigate the node tree using node relationships.

```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>

  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```



DOM navigation

```
<html>
<body>

<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
// 1()
document.getElementById("id02").innerHTML = document.getElementById("id01").innerHTML;
// (2)
document.getElementById("id02").innerHTML =
           document.getElementById("id01").firstChild.nodeValue;
// (3)
document.getElementById("id02").innerHTML =
           document.getElementById("id01").childNodes[0].nodeValue;
</script>

</body>
</html>
```

nodeName Property

- The nodeName property specifies the name of a node.
  - nodeName is read-only
  - nodeName of an **element node** is the same as the tag name
  - nodeName of an **attribute node** is the attribute name
  - nodeName of a **text node** is always #text
  - nodeName of the **document node** is always #document
  - nodeName always contains the uppercase tag name of an HTML element.

```
<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").nodeName;
</script>
```

nodeValue Property

- The nodeValue property specifies the value of a node.
  - nodeValue for **element nodes** is null
  - nodeValue for **text nodes** is the text itself
  - nodeValue for **attribute nodes** is the attribute value

nodeType Property

- The nodeType property is read only. It returns the type of a node.

```
<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").nodeType;
</script>
```

Node	Type	Example
ELEMENT_NODE	1	<h1 class="heading">W3Schools</h1>
ATTRIBUTE_NODE	2	class = "heading" (deprecated)
TEXT_NODE	3	W3Schools
COMMENT_NODE	8	<!-- This is a comment -->
DOCUMENT_NODE	9	The HTML document itself (the parent of <html>)
DOCUMENT_TYPE_NODE	10	<!Doctype html>

DOM Nodes

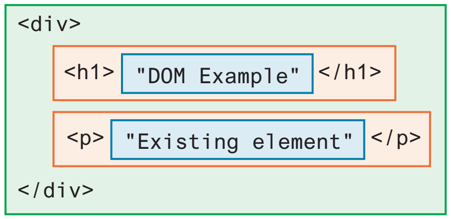
Property	Description
attributes	Collection of node attributes
childNodes	A NodeList of child nodes for this node
firstChild	First child node of this node.
lastChild	Last child of this node.
nextSibling	Next sibling node for this node.
nodeName	Name of the node
nodeType	Type of the node
nodeValue	Value of the node
parentNode	Parent node for this node.
previousSibling	Previous sibling node for this node.

Creating New HTML Elements

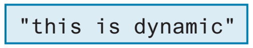
- To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

```
<div id="first">
    <h1>DOM Example</h1>
    <p>Existing element</p>
</div>
```

Visualizing the DOM elements

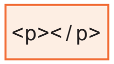


1 Create a new text node



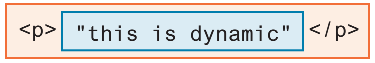
```
var text = document.createTextNode("this is dynamic");
```

2 Create a new empty <p> element



```
var p = document.createElement("p");
```

3 Add the text node to new <p> element

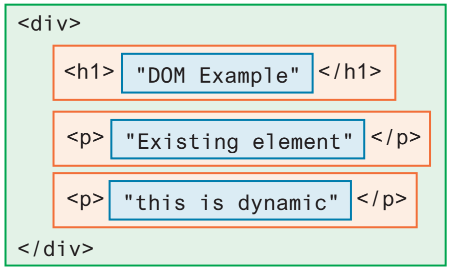


```
p.appendChild(text);
```

4 Add the <p> element to the <div>

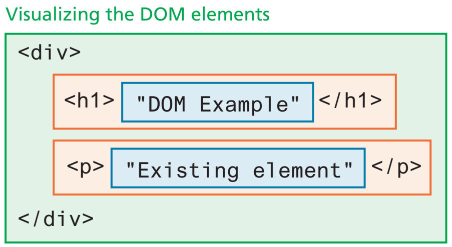
```
var first = document.getElementById("first");
first.appendChild(p);
```

```
<div id="first">
    <h1>DOM Example</h1>
    <p>Existing element</p>
    <p>this is dynamic</p>
</div>
```

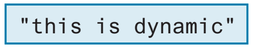


Creating New HTML Elements

```
<div id="first">
    <h1>DOM Example</h1>
    <p>Existing element</p>
</div>
```

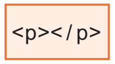


1 Create a new text node



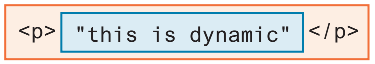
```
var text = document.createTextNode("this is dynamic");
```

2 Create a new empty <p> element



```
var p = document.createElement("p");
```

3 Add the text node to new <p> element

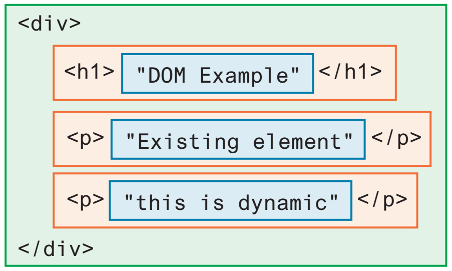


```
p.appendChild(text);
```

4 Add the <p> element to the <div>

```
var first = document.getElementById("first");
first.appendChild(p);
```

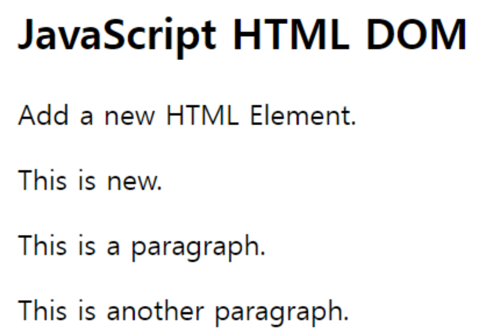
```
<div id="first">
    <h1>DOM Example</h1>
    <p>Existing element</p>
    <p>this is dynamic</p>
</div>
```



Creating new HTML Elements

- The *appendChild()* method appended the new element as the last child of the parent.
- If you don't want that you can use the *insertBefore()* method:

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);

const element = document.getElementById("div1");
const child = document.getElementById("p1");
element.insertBefore(para, child);
</script>
```



JavaScript HTML DOM

Add a new HTML Element.

This is new.

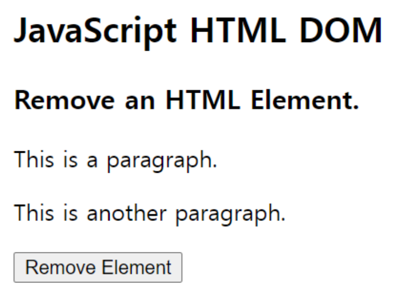
This is a paragraph.

This is another paragraph.

Removing HTML Elements

- To remove an HTML element, use the remove() method:

```html
<h2>JavaScript HTML DOM</h2>
<h3>Remove an HTML Element.</h3>

<div>
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<button onclick="myFunction()">Remove Element</button>

<script>
function myFunction() {
document.getElementById("p1").remove();
}
</script>
```

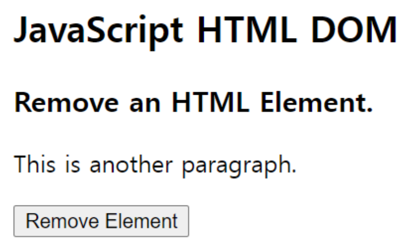


JavaScript HTML DOM

Remove an HTML Element.

This is a paragraph.

This is another paragraph.



JavaScript HTML DOM

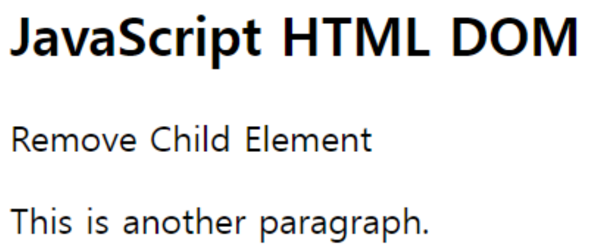
Remove an HTML Element.

This is another paragraph.

Removing a Child Node

- For browsers that does not support the remove() method, you have to find the parent node to remove an element:

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
const parent =
document.getElementById("div1");
const child = document.getElementById("p1");
parent.removeChild(child);
</script>
```



JavaScript HTML DOM

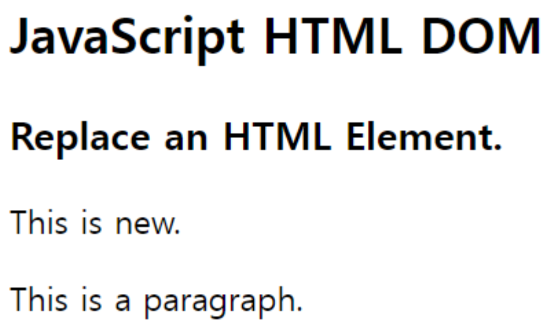
Remove Child Element

This is another paragraph.

Replacing HTML Elements

- To replace an element to the HTML DOM, use the replaceChild() method:

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
const para = document.createElement("p");
const node = document.createTextNode("This
is new.");
para.appendChild(node);

const parent =
document.getElementById("div1");
const child = document.getElementById("p1");
parent.replaceChild(para, child);
</script>
```



JavaScript HTML DOM

Replace an HTML Element.

This is new.

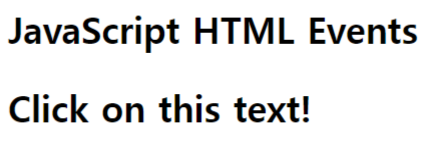
This is a paragraph.

Event handling

Reacting to Events

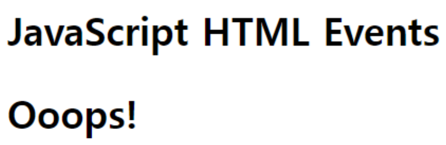
- A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.
- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:
  - onclick=*JavaScript*

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

<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>

</body>
</html>
```



JavaScript HTML Events

Click on this text!



JavaScript HTML Events

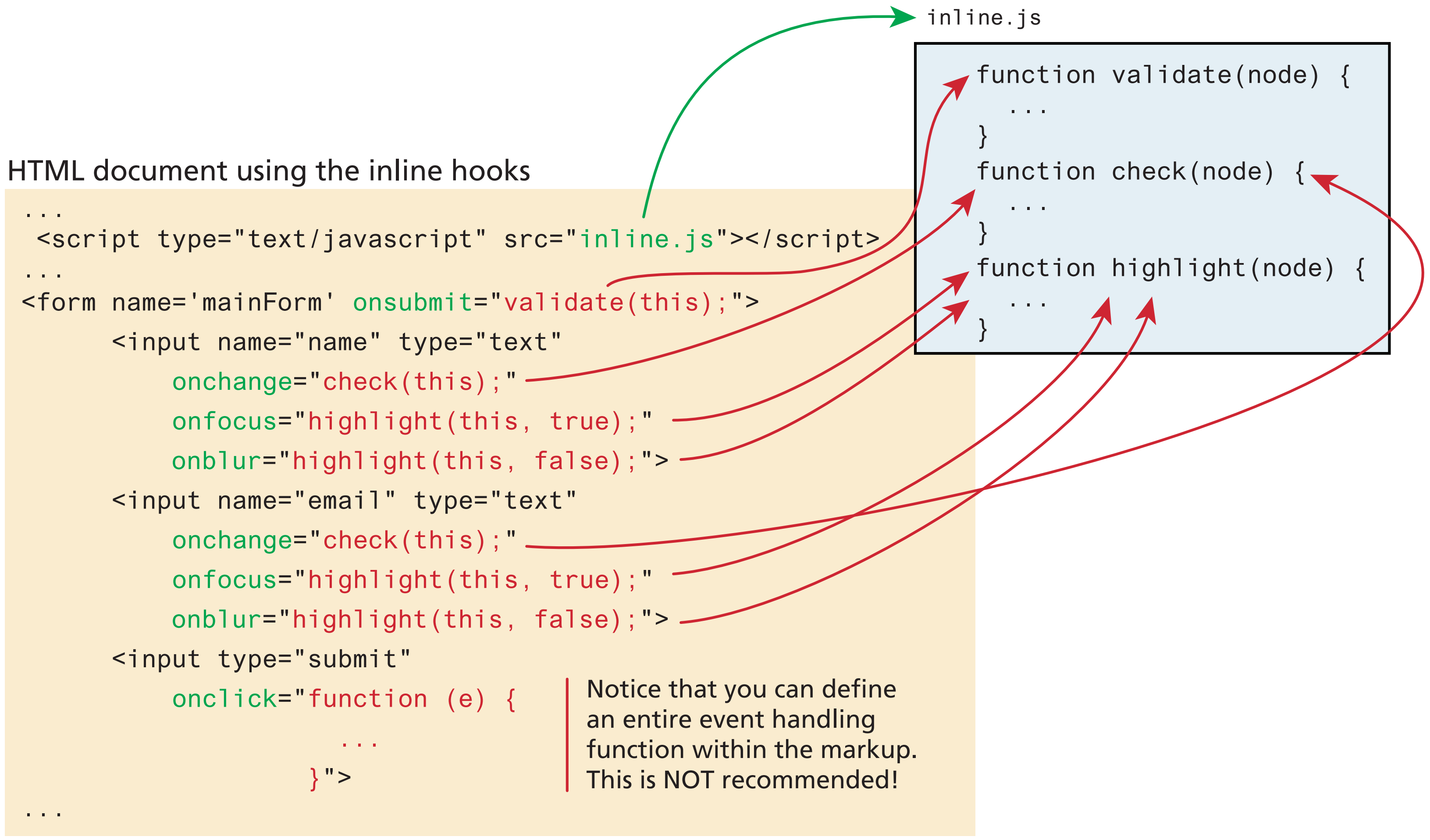
Ooops!

Reacting to Events

- In this example, a function is called from the event handler:

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

<h1 onclick="changeText(this)">Click on this text!</h1>

<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>

</body>
</html>
```

Inline Hooks



HTML document using the inline hooks

```
...
 <script type="text/javascript" src="inline.js"></script>
...
<form name='mainForm' onsubmit="validate(this);">
      <input name="name" type="text"
          onchange="check(this);"
          onfocus="highlight(this, true);"
          onblur="highlight(this, false);">
      <input name="email" type="text"
          onchange="check(this);"
          onfocus="highlight(this, true);"
          onblur="highlight(this, false);">
      <input type="submit"
          onclick="function (e) {
                      ...
                 }">
...
```

inline.js

```
function validate(node) {
   ...
}
function check(node) {
   ...
}
function highlight(node) {
   ...
}
```

Notice that you can define an entire event handling function within the markup. This is NOT recommended!

Assign Events Using HTML DOM

- The HTML DOM allows you to assign events to HTML elements using JavaScript:

```
<body>

  <h2>JavaScript HTML Events</h2>
  <p>Click "Try it" to execute the displayDate() function.</p>

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

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

  <script>
  document.getElementById("myBtn").onclick = displayDate;

  function displayDate() {
    document.getElementById("demo").innerHTML = Date();
  }
  </script>

  </body>
```

HTML DOM EventListener

- The *addEventListener()* method attaches an event handler to the specified element.
- The *addEventListener()* method attaches an event handler to an element without overwriting existing event handlers.
  - You can add many event handlers to one element.
  - You can add many event handlers of the same type to one element, i.e two "click" events.
- When using the *addEventListener()* method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

```
element.addEventListener(event, function, useCapture);
element.removeEventListener(event, function);
```

HTML DOM EventListener

```
<body>

  <h2>JavaScript addEventListener()</h2>

  <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>

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

  function myFunction() {
    alert ("Hello World!");
  }
  </script>

  </body>
```

HTML DOM EventListener

- Add Many Event Handlers to the Same Element

```
<body>

  <h2>JavaScript addEventListener()</h2>

  <p>This example uses the addEventListener() method to add two click events
to the same button.</p>

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

  <script>
  var x = document.getElementById("myBtn");
  x.addEventListener("click", myFunction);
  x.addEventListener("click", someOtherFunction);

  function myFunction() {
    alert ("Hello World!");
  }

  function someOtherFunction() {
    alert ("This function was also executed!");
  }
  </script>

  </body>
```

HTML DOM EventListener

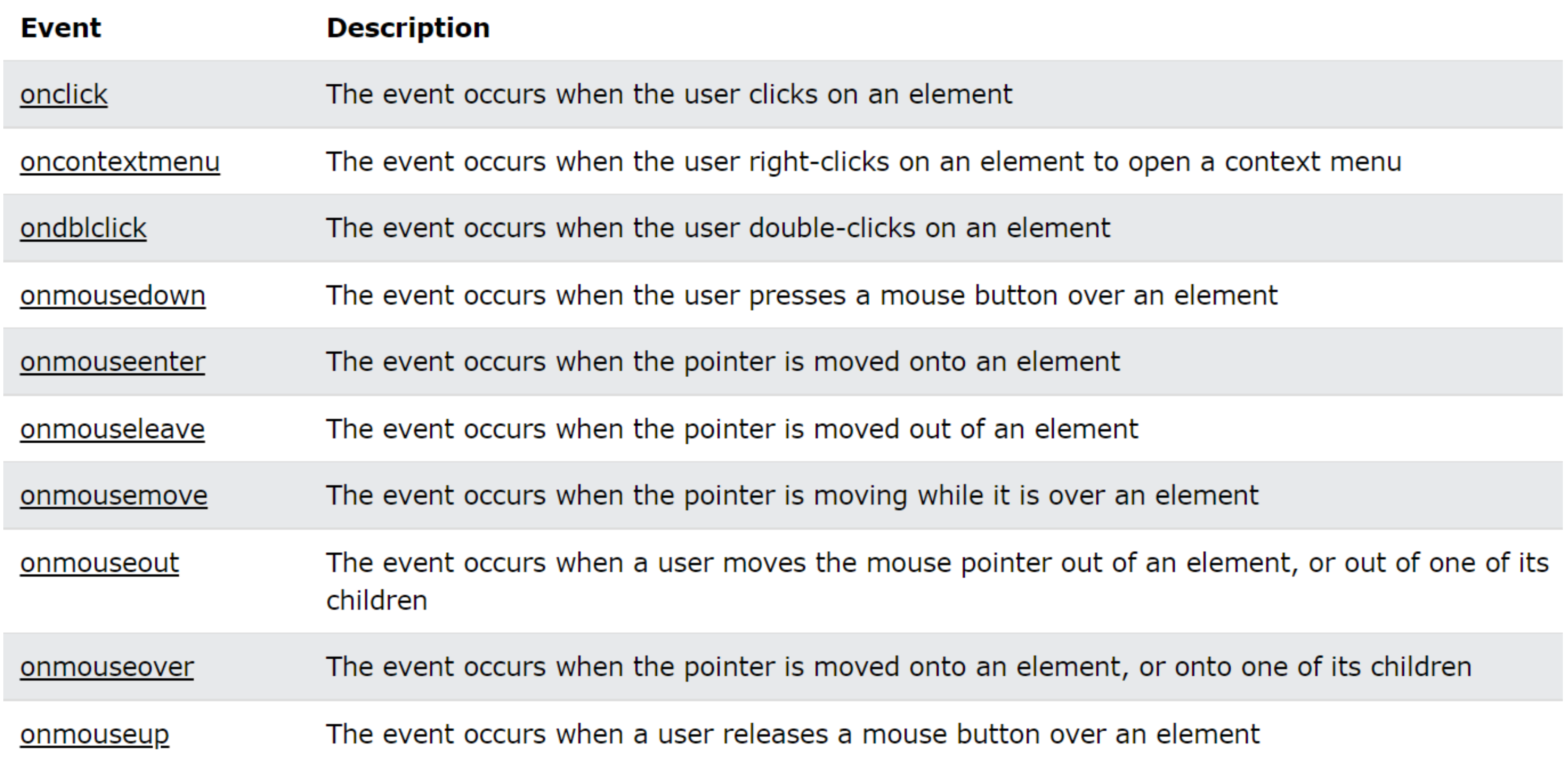
- Passing Parameters

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

```
  <h2>JavaScript addEventListener()</h2>
  <p>This example demonstrates how to pass parameter values when using the
addEventListener() method.</p>
  <p>Click the button to perform a calculation.</p>

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

  <script>
  let p1 = 5;
  let p2 = 7;
  document.getElementById("myBtn").addEventListener("click", function() {
    myFunction(p1, p2);
  });

  function myFunction(a, b) {
    document.getElementById("demo").innerHTML = a * b;
  }
  </script>
```

Event Types

- Mouse Events



Event	Description
onclick	The event occurs when the user clicks on an element
oncontextmenu	The event occurs when the user right-clicks on an element to open a context menu
ondblclick	The event occurs when the user double-clicks on an element
onmousedown	The event occurs when the user presses a mouse button over an element
onmouseenter	The event occurs when the pointer is moved onto an element
onmouseleave	The event occurs when the pointer is moved out of an element
onmousemove	The event occurs when the pointer is moving while it is over an element
onmouseout	The event occurs when a user moves the mouse pointer out of an element, or out of one of its children
onmouseover	The event occurs when the pointer is moved onto an element, or onto one of its children
onmouseup	The event occurs when a user releases a mouse button over an element

Event Types

- Keyboard Events / Form Events

Attribute	Value	Description
onkeydown	*script*	Fires when a user is pressing a key
onkeypress	*script*	Fires when a user presses a key
onkeyup	*script*	Fires when a user releases a key

Attribute	Value	Description
onblur	*script*	Fires the moment that the element loses focus
onchange	*script*	Fires the moment when the value of the element is changed
oncontextmenu	*script*	Script to be run when a context menu is triggered
onfocus	*script*	Fires the moment when the element gets focus
oninput	*script*	Script to be run when an element gets user input
oninvalid	*script*	Script to be run when an element is invalid
onreset	*script*	Fires when the Reset button in a form is clicked
onsearch	*script*	Fires when the user writes something in a search field (for <input="search">)
onselect	*script*	Fires after some text has been selected in an element
onsubmit	*script*	Fires when a form is submitted

Event Objects

- When an event is triggered, the browser will construct an event object that contains information about the event.

```
<body>

  <div onmousemove="showCoords(event)" onmouseout="clearCoor()"></div>

  <p>Mouse over the rectangle above to get the horizontal and vertical 
coordinates of your mouse pointer.</p>

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

  <script>
  function showCoords(event) {
    var x = event.clientX;
    var y = event.clientY;
    var coor = "X coords: " + x + ", Y coords: " + y;
    document.getElementById("demo").innerHTML = coor;
  }

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

  </body>
```

Event Objects

- KeyboardEvent Object

```
<body>

  <p>Press the "O" key on the keyboard in the input field to
alert some text.</p>

  <input type="text" size="40" onkeypress="myFunction(event)">

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

  <script>
  function myFunction(event) {
    var x = event.charCode || event.keyCode;
    if (x == 111 || x == 79) {  // o is 111, O is 79
      alert("You pressed the 'O' key!");
    }
  }
  </script>

  </body>
```

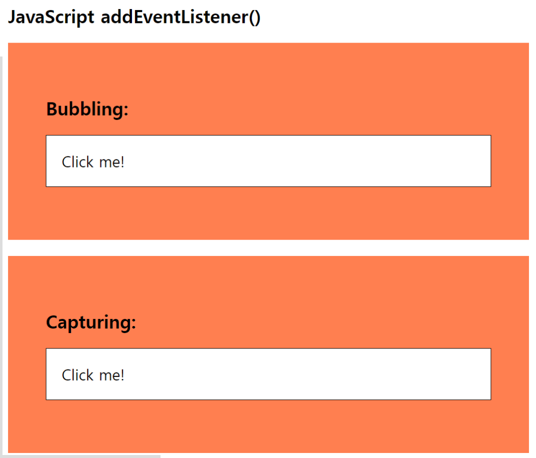
Event Bubbling/Capturing

- Event propagation is a way of defining the element order when an event occurs.

- If you have a **<p>** element inside a **<div>** element, and the user clicks on the **<p>** element, which element's "click" event should be handled first?
- In *bubbling* the **inner most element's event is handled first** and then the outer: the <p> element's click event is handled first, then the <div> element's click event.
- In *capturing* the **outer most element's event is handled first** and then the inner: the <div> element's click event will be handled first, then the <p> element's click event.

Event Bubbling/Capturing

```
<h2>JavaScript addEventListener()</h2>

<div id="myDiv1">
  <h2>Bubbling:</h2>
  <p id="myP1">Click me!</p>
</div><br>
<div id="myDiv2">
  <h2>Capturing:</h2>
  <p id="myP2">Click me!</p>
</div>

<script>
document.getElementById("myP1").addEventListener("click", function() {
  alert("You clicked the white element!");
}, false);

document.getElementById("myDiv1").addEventListener("click", function() {
  alert("You clicked the orange element!");
}, false);

document.getElementById("myP2").addEventListener("click", function() {
  alert("You clicked the white element!");
}, true);

document.getElementById("myDiv2").addEventListener("click", function() {
  alert("You clicked the orange element!");
}, true);
</script>
```


JavaScript addEventListener()

Bubbling:

Click me!

Capturing:

Click me!

Form Validation

JavaScript Form Validation

- The function can be called when the form is submitted:

```
<head>
  <script>
  function validateForm() {
    let x = document.forms["myForm"]["fname"].value;
    if (x == "") {
      alert("Name must be filled out");
      return false;
    }
  }
  </script>
  </head>
  <body>

  <h2>JavaScript Validation</h2>

  <form name="myForm" action="/action_page.php" onsubmit="return validateForm()"
method="post">
    Name: <input type="text" name="fname">
    <input type="submit" value="Submit">
  </form>

  </body>
```

Automatic HTML Form Validation

- HTML form validation can be performed automatically by the browser:
- If a form field (fname) is empty, the required attribute prevents this form from being submitted:

```
<body>

  <h2>JavaScript Validation</h2>

  <form action="/action_page.php" method="post">
    <input type="text" name="fname" required>
    <input type="submit" value="Submit">
  </form>

  <p>If you click submit, without filling out the text field,
  your browser will display an error message.</p>

  </body>
```