

Client-side Technologies

Eng. Niveen Nasr El-Den
SD & Gaming CoE
iTi

Day 8

JavaScript Built-in Objects

- String
- Number
- Array
- Date
- Math
- Boolean
- RegExp
- Error
- Function
- Object

Error built-in Objects

Error Object Creation

- Whenever an error occurs, an instance of **error** object is created to describe the error.
- Error objects are created either by the environment (the browser) or by your code.
- Developer can create Error objects by 2 ways:
 - ▷ **Explicitly:**
 - `var newErrorObj = new Error();`
 - ▷ **Implicitly:**
 - thrown using the throw statement

Error Object Construction

- Error constructor
 - ▷ `var e = new Error();`
- Six additional Error constructor ones exist and they all inherit Error:

EvalError	Raised by eval when used incorrectly
RangeError	Numeric value exceeds its range
ReferenceError	Invalid reference is used
SyntaxError	Used with invalid syntax
TypeError	Raised when variable is not the type expected
URIError	Raised when <code>encodeURIComponent()</code> or <code>decodeURIComponent()</code> are used incorrectly

- Using *instanceOf* when catching the error lets you know if the error is one of these built-in types.

Error Object Properties

Property	Description
description	Plain-language description of error (IE only)
fileName	URI of the file containing the script throwing the error
lineNumber	Source code line number of error
message	Plain-language description of error (ECMA)
name	Error type (ECMA)
number	Microsoft proprietary error number

Error Object Standard Properties

- **name** → The name of the error constructor used to create the object

▷ Example:

- `var e = new EvalError('Oops');`
- `e.name;`

→ "EvalError"

- **Message** → Additional error information:

▷ Example:

- `var e = new Error('jaavcsritp is _not_ how you spell it');`
- `e.message`

→ "jaavcsritp is _not_ how you spell it"

Example!

Error Handling

JavaScript Error Handling

- There are two ways of catching errors in a Web page:
 1. *try...catch* statement.
 2. *onerror* event.

try...catch Statement

- The try...catch statement allows you to test a block of code for errors.
- The **try** *block* contains the code to be run.
- The **catch** *block* contains the code to be executed if an error occurs.
- Syntax

try

{

//Run some code here

}

catch(err)



Implicitly an Error
object "err" is created

{

//Handle errors here

}

try...catch Statement

try {

✓ no error.

✓ no error.

an error! *control is passed to the catch block here.*

this will never execute.

}

catch(exception)

{

✓ error handling code is run here

}

✓ execution continues from here.

Example!

throw Statement

- The throw statement allows you to create an exception.
- Using throw statement with the try...catch, you can control program flow and generate accurate error messages.
- **Syntax**
`throw(exception)`
- The exception can be a **string**, **integer**, **Boolean** or an **object**

try...catch & throw Example

```
try{  
    if(x<100)  
        throw "less100"  
    else if(x>200)  
        throw "more200"  
}  
catch(er){  
    if(er=="less100")  
        alert("Error! The value is too low")  
    if(er == "more200")  
        alert("Error! The value is too high")  
}
```

Example!

Adding the *finally* statement

- If you have any functionality that needs to be processed regardless of success or failure, you can include this in the *finally* block.

try...catch...finally Statement

try {

- ✓ no error.
- ✓ no error.
- ✓ no error.

}

catch(exception)

{

- ✓ ~~error handling code will not run.~~

}

finally {

- ✓ This code will run even there is no failure occurrence.

}

- ✓ execution will be continued.



try...catch...finally Statement

try {

- ✓ no error.
- ✓ no error.

an error! *control is passed to the catch block here.*
this will never execute.

}

catch(exception)

{

- ✓ error handling code is run here
- ✓ error handling code is run here
- ✓ error handling code is run here

}

finally {

- ✓ This code will run even there is failure occurrence.

}

- ✓ execution will be continued.

Example!

try...catch...finally Statement

try {

- ✓ no error.
- ✓ no error.

an error! *control is passed to the catch block here.*
this will never execute.

}

catch(exception)

{

- ✓ error handling code is run here

an error!

~~error handling code is run here will never execute.~~

}

finally {

- ✓ This code will run even there is failure occurrence.

}

~~execution wont be continued.~~

Example!

onerror Event

- The old standard solution to catch errors in a web page.
- The *onerror* event is fired whenever there is a script error in the page.
- onerror event can be used to:
 - ▷ Suppress error.
 - ▷ Retrieve additional information about the error.

Suppress error

```
function supError()  
{  
    alert("Error occurred")  
}  
window.onerror=supError
```

OR

```
function supError()  
{  
    return true; //or false;  
}  
window.onerror=supError
```

The value returned determines whether the browser displays a standard error message.

true the browser does **not** display the standard error message.

false the browser **displays** the standard error message in the JavaScript console

Retrieve additional information about the error

onerror=handleErr

```
function handleErr(msg,url,l,col,err)
{
    //Handle the error here
    return true; //or false;
}
```

where

- msg → Contains the message explaining why the error occurred.
- url → Contains the url of the page with the error script
- l → Contains the line number where the error occurred
- col → Column number for the line where the error occurred
- err → Contains the error object

Document Object Model

DOM

DOM

- DOM Stands for Document Object Model.
- W3C standard.
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
- Its an API that interact with documents like HTML, XML.. etc.
- Defines a standard way to access and manipulate HTML documents.
- Platform independent.
- Language independent

DOM

- The **document** object in the **BOM** is the top level of the **DOM** hierarchy.
- DOM is a representation of the whole document as nodes and attributes.
- You can access each of these nodes and attributes and change or remove them.
- You can also create new ones or add attributes to existing ones.

DOM is a **subset** of **BOM**.

In other word: **the document is yours!**

DOM Relationships

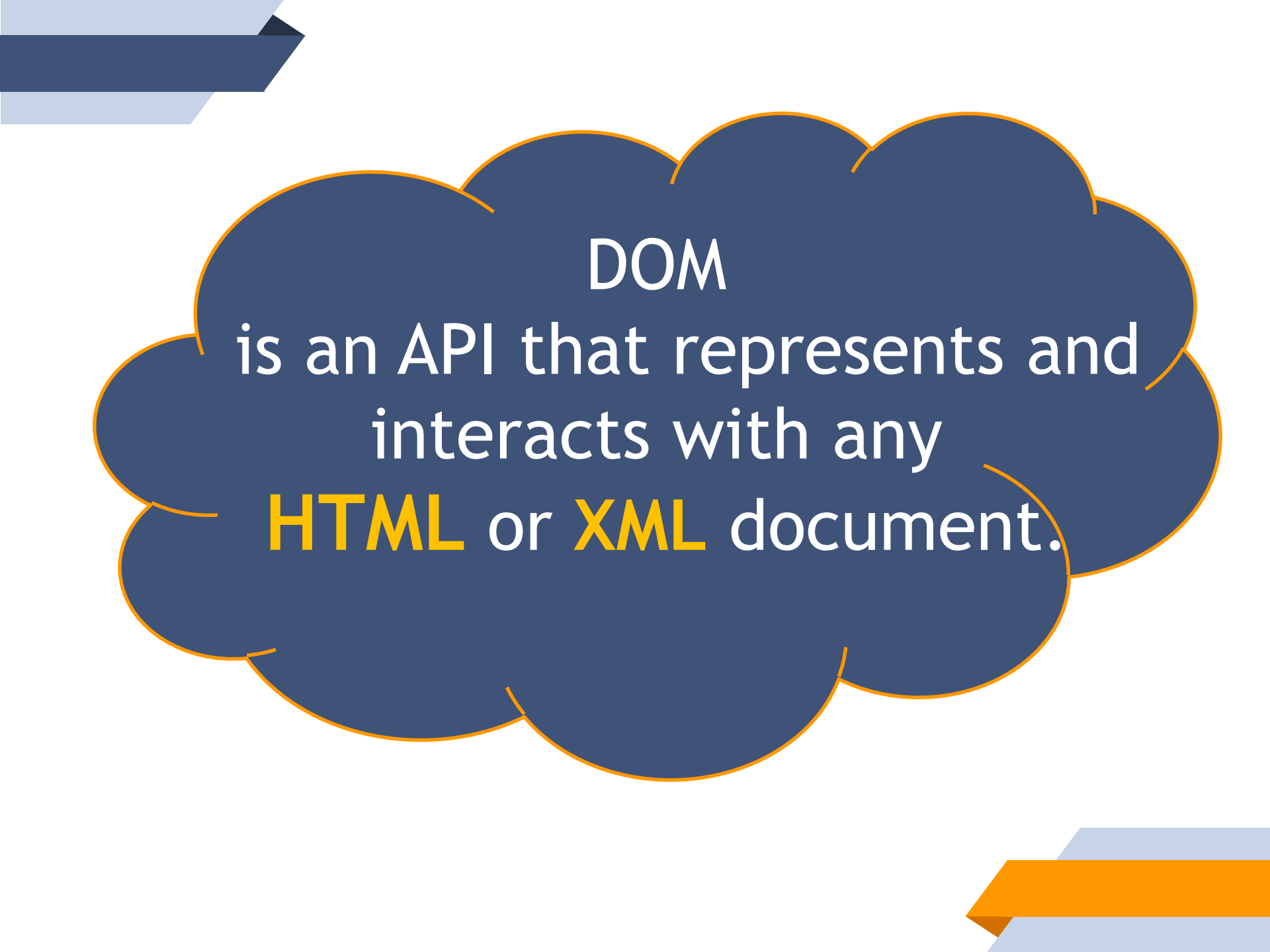
Scripting HTML

HTML DOM

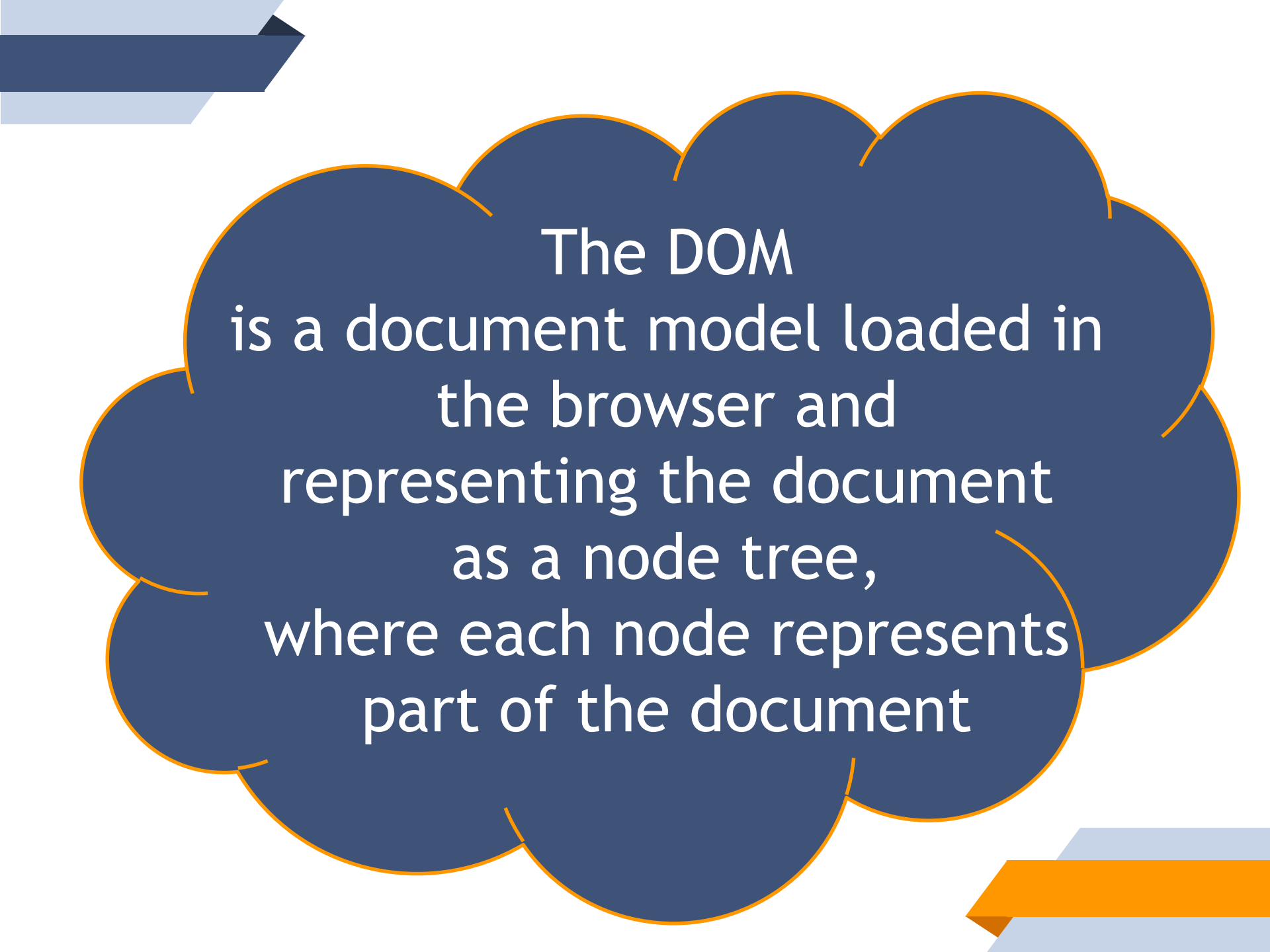
- The HTML DOM is a standard for how to **get**, **change**, **add**, or **delete** HTML elements.
- It is a hierarchy of data types for HTML documents, links, forms, comments, and everything else that can be represented in HTML code.
- The general data type for objects in the DOM are **Nodes**. They have **attributes**, and some nodes can contain other nodes.
- There are several node types, which represent more specific data types for HTML elements.
Node types are represented by numeric constants.

DOM

- It allows code running in a browser to access and interact with every node in the document.
- Nodes can be created, moved and changed.
- Event listeners can be added to nodes and triggered on occurrence of a given event.



DOM
is an API that represents and
interacts with any
HTML or **XML** document.




The DOM
is a document model loaded in
the browser and
representing the document
as a node tree,
where each node represents
part of the document

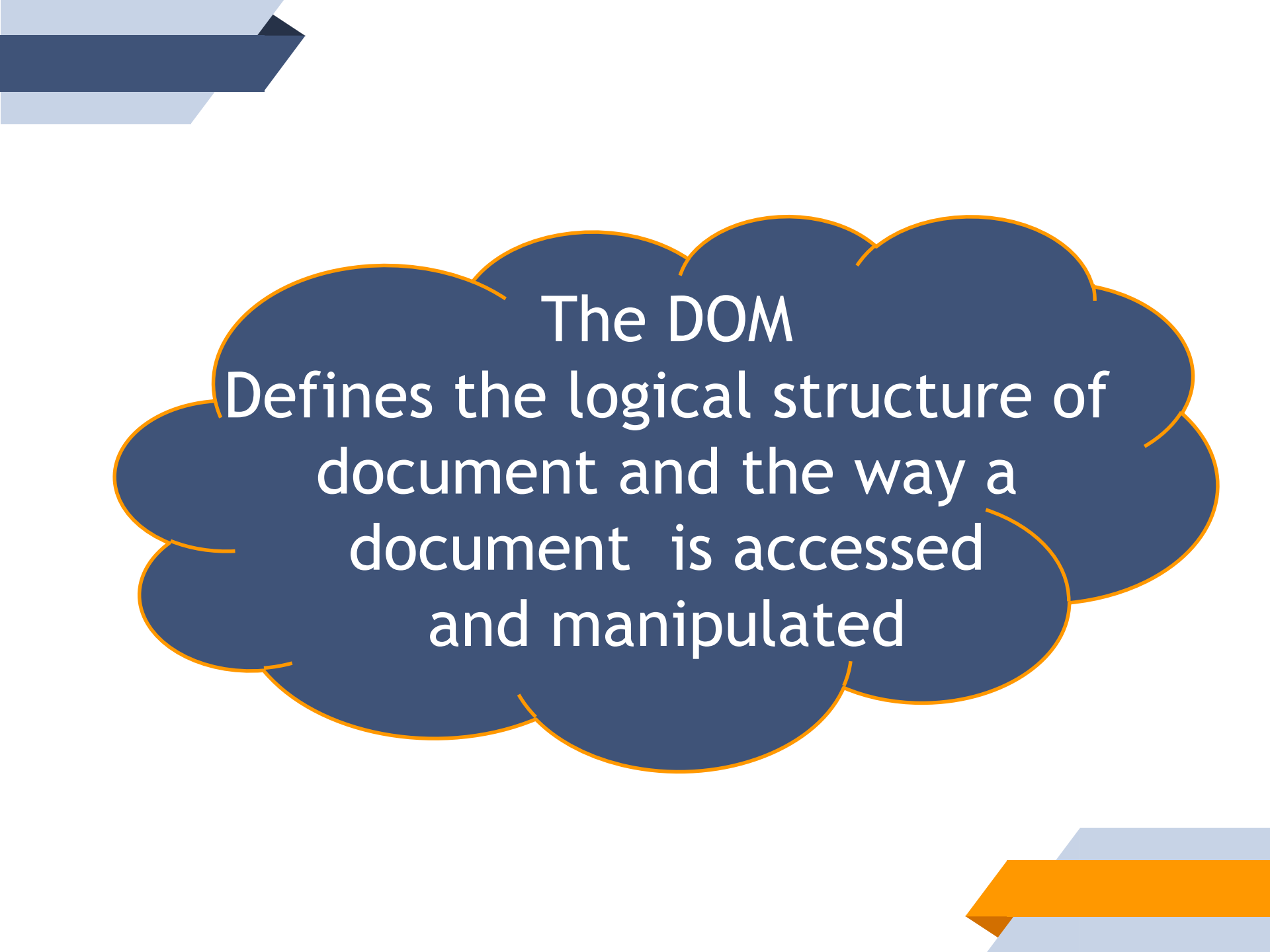


The DOM
is an application programming
interface “API”



a set of functions or
methods used to access
some functionality



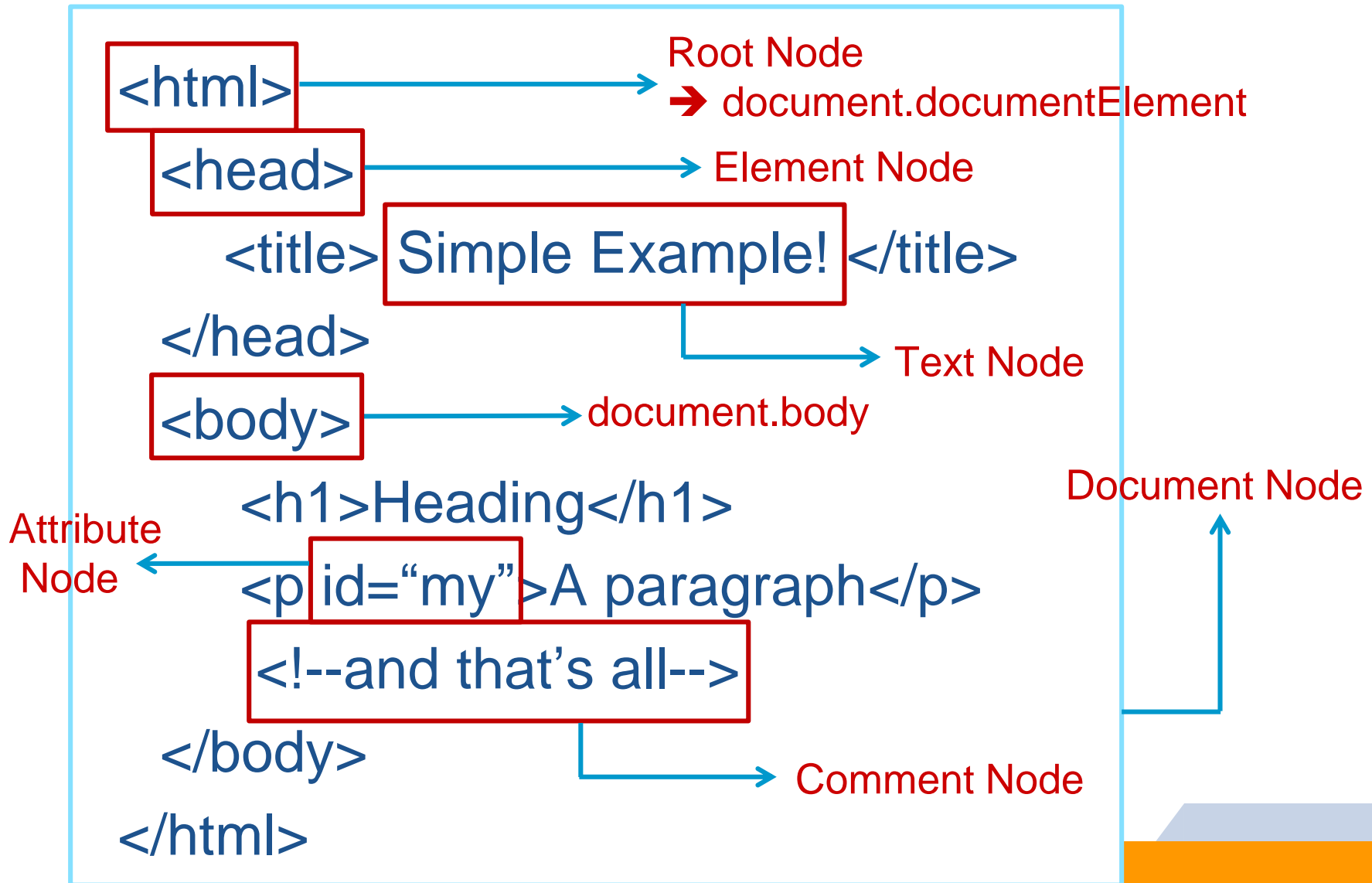


The DOM
Defines the logical structure of
document and the way a
document is accessed
and manipulated

HTML DOM

- According to the DOM, everything in an HTML document is a node.
- The DOM says:
 - ▷ The entire document is a **document node**
 - ▷ Every HTML element is an **element node**
 - ▷ The text in the HTML elements are **text nodes**
 - ▷ Every HTML attribute is an **attribute node**
 - ▷ Comments are **comment nodes**
- JavaScript is powerful DOM Manipulation

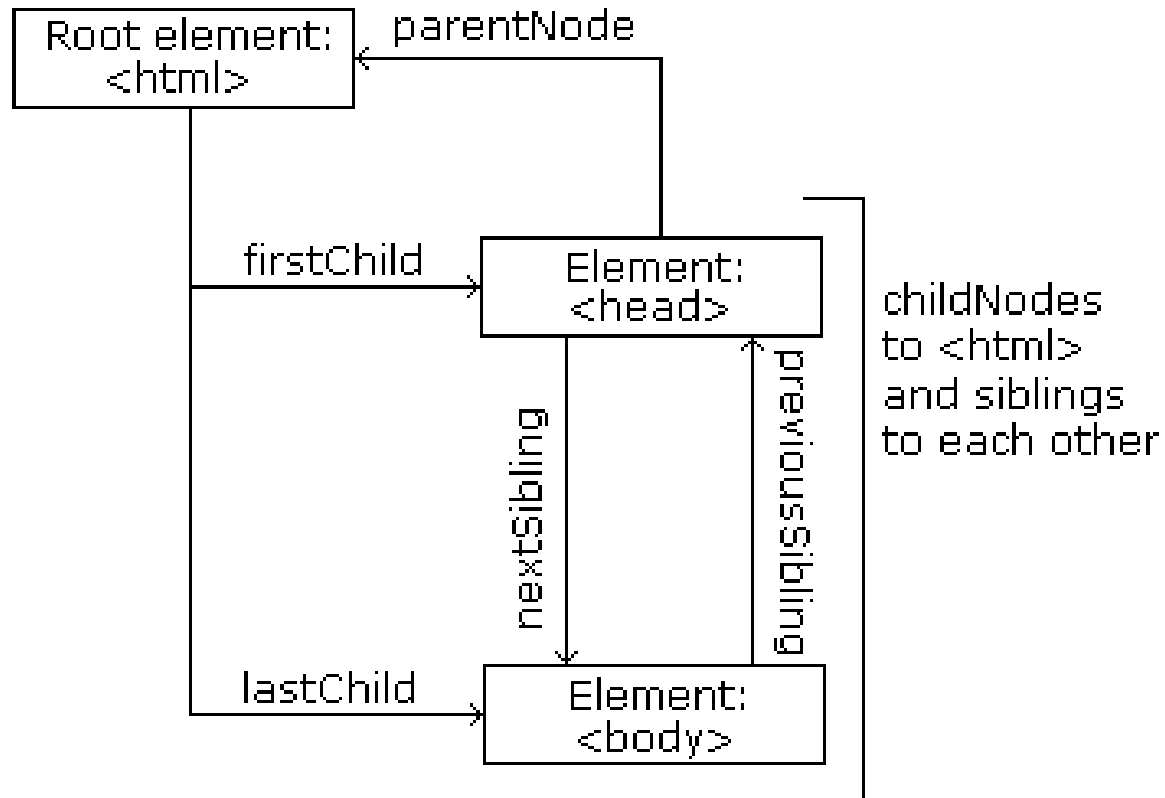
Simple Example!



Node Tree

- The HTML DOM views HTML document as a node-tree.
- All the nodes in the tree have relationships to each other.

- ▷ Parent
 - parentNode
- ▷ Children
 - firstChild
 - lastChild
- ▷ Sibling
 - nextSibling
 - previousSibling

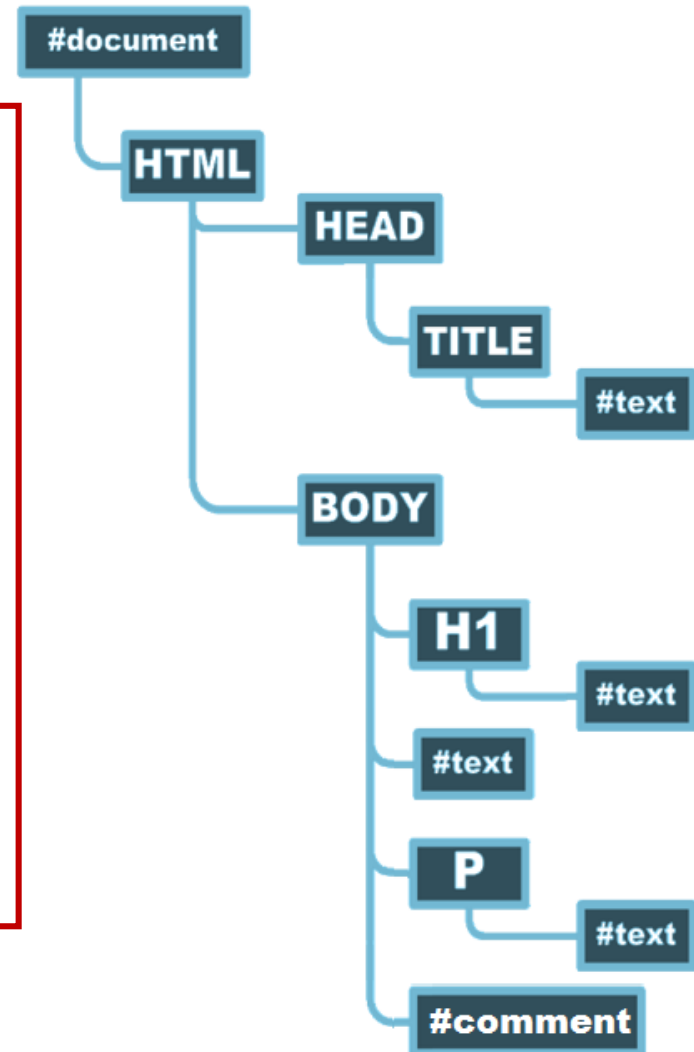


Nodes Relationships

- The terms **parent**, **child**, and **sibling** are used to describe the relationships.
 - ▷ Parent nodes have children.
 - ▷ Children on the same level are called siblings (brothers or sisters).
- **Attribute** nodes are not child nodes of the element they belong to, and have **no parent** or **sibling** nodes
- In a node tree, the top node is called the **root**
- Every node, except the root, has exactly one **parent** node
- A node can have any number of **children**
- A **leaf** is a node with no children
- **Siblings** are nodes with the same parent

Simple Example!

```
<html>
  <head>
    <title>Simple Example!</title>
  </head>
  <body>
    <h1>Greeting</h1>
    Welcome All
    <p>A paragraph</p>
    <!-- and that's all-->
  </body>
</html>
```



Node Properties

- All nodes have three main properties

Property	Description
<i>nodeName</i>	Returns HTML Tag name in uppercase display
<i>tagname</i>	
<i>nodeType</i>	returns a numeric constant to determine node type. There are 12 node types.
<i>nodeValue</i>	returns null for all node types except for text and comment nodes.

Using **nodeName**

If node is text it returns **#text**

For comment it returns

#comment

For document it returns

#document

Value	Description
1	Element Node
2	Attribute Node
3	Text Node
8	Comment Node
9	Document Node

To get the Root Element:
document.documentElement.

Node Collections

- Node Collections have One Property
 - ▷ **length** : gives the length of the Collection.
 - e.g. `childNodes.length`: returns number of elements inside the collection
- We can check if there is child collection using
 - ▷ **hasChildNodes()**: Tells if a node has any children
- We can check if there is attribute collection using
 - ▷ **hasAttributes()**: Tells if a node has any attributes

Collection	Description	Accessing
<code>childNodes</code>	Collection of element's children	<code>childNodes[]</code> <code>childNodes.item()</code>
<code>attributes</code>	Returns collection of the attributes of an element	<code>attributes[]</code> <code>attributes.item()</code>

Dealing With Nodes

- Dealing with nodes fall into four main categories:
 - ▷ Accessing Node
 - ▷ Modifying Node's content
 - ▷ Adding New Node
 - ▷ Remove Node from tree

Accessing DOM Nodes

- You can access a node in **5** main ways:
 - ▷ [window.]document.**getElementById**("id")
 - ▷ [window.]document.**getElementsByName**("name")
 - ▷ [window.]document.**getElementsByTagName**("tagname")
 - ▷ By navigating the node tree, using the node relationships
 - ▷ New HTML5 Selectors.

Example!

New HTML5 Selectors

- In HTML5 we can select elements by ClassName

```
var elements = document.getElementsByClassName('entry');
```

- Moreover there's now possibility to fetch elements that match provided CSS syntax

```
var elements = document.querySelectorAll(".someClasses");
```

```
var elements = document.querySelectorAll("div,p");
```

```
var elements = document.querySelector("#someID");
```

```
var first_td = document.querySelector("span");
```

Accessing DOM Nodes

Navigating the node tree, using the node relationships

firstChild	Move direct to first child
lastChild	Move direct to last child
parentNode	To access child's parent
nextSibling	Navigate down the tree one node step
previousSibling	Navigate up the tree one node step
Using children collection → <code>childNodes[]</code>	

Example!

Modifying Node's Content

- Changing the Text Node by using

innerHTML	Sets or returns the HTML contents (+text) of an element
textContent	Equivalent to innerText.
nodeValue → with text and comment nodes only	
setAttribute()	Modify/Adds a new attribute to an element
just using attributes as object properties	

- Modifying Styles
 - ▷ Node.style

Example!

Node's Class Attribute

- The global **class** attribute is get and set via **className** property
- The **classList** property returns a collection of the class attributes of the caller element, it has the following methods
 - ▷ `add("classNm")`
 - ▷ `remove("classNm")`
 - ▷ `toggle("classNm")`
 - ▷ `replace("oldClassNm", "newClassNm")`

Creating & Adding Nodes

Method	Description
createElement()	To create new tag element
createTextNode()	To create new text element
createAttribute()	To creates an attribute element
createComment()	To creates an comment element

Creating & Adding Nodes

Method	Description
cloneNode (true false)	Creating new node a copy of existing node. It takes a Boolean value true : Deep copy with all its children or false : Shallow copy only the node
b.appendChild (a)	To add new created node “a” to DOM Tree at the end of the selected element “b”.
insertBefore (a,b)	Similar to appendChild() with extra parameter, specifying before which element to insert the new node. a: the node to be inserted b: where a should be inserted before document.body.insertBefore(a,b)

Example!

Removing DOM Nodes

Method	Description
<code>removeChild()</code>	To remove node from DOM tree
<code>parent.replaceChild(n,o)</code>	To remove node from DOM tree and put another one in its place n: new child o: old child
<code>removeAttribute()</code>	Removes a specified attribute from an element

- A quick way to wipe out all the content of a subtree is to set the `innerHTML` to a blank string. This will remove all of the children of `<body>`

```
document.body.innerHTML="";
```

Example!

Summary

- Access nodes:
 - ▷ Using parent/child relationship properties parentNode, childNodes, firstChild, lastChild, nextSibling, previousSibling
 - ▷ Using getElementById(), getElementsByTagName(), getElementByName()
- Modify nodes:
 - ▷ Using innerHTML or innerText/textContent
 - ▷ Using nodeValue or setAttribute() or just using attributes as object properties
- Remove nodes with
 - ▷ removeChild() or replaceChild()
- And add new ones with
 - ▷ appendChild(), cloneNode(), insertBefore()

Dynamic HTML

*the art of making dynamic and interactive
web pages.*

DHTML

- DHTML has no official definition or specification.
- DHTML stands for Dynamic HTML.
- DHTML is NOT a scripting language.
- DHTML is not w3c (i.e. not a standard).
- DHTML is a browser feature-that gives you the ability to make dynamic Web pages.
- "***Dynamic***" is defined as the ability of the browser to alter a web page's look and style *after* the document has been loaded.
- DHTML is very important in web development

DHTML

- DHTML uses a combination of:

1. Scripting language
2. DOM
3. CSS

to create HTML that can change even after a page has been loaded into a browser.

- DHTML is supported by 4.x generation browsers.

Assignment