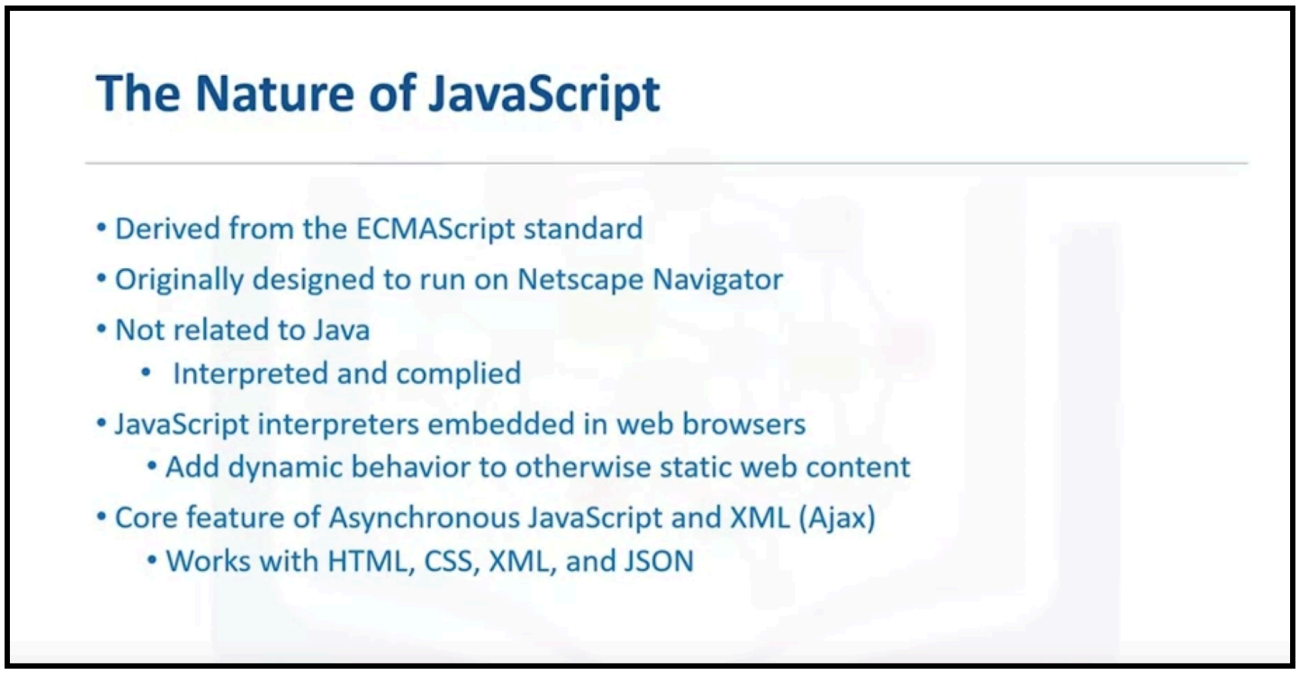
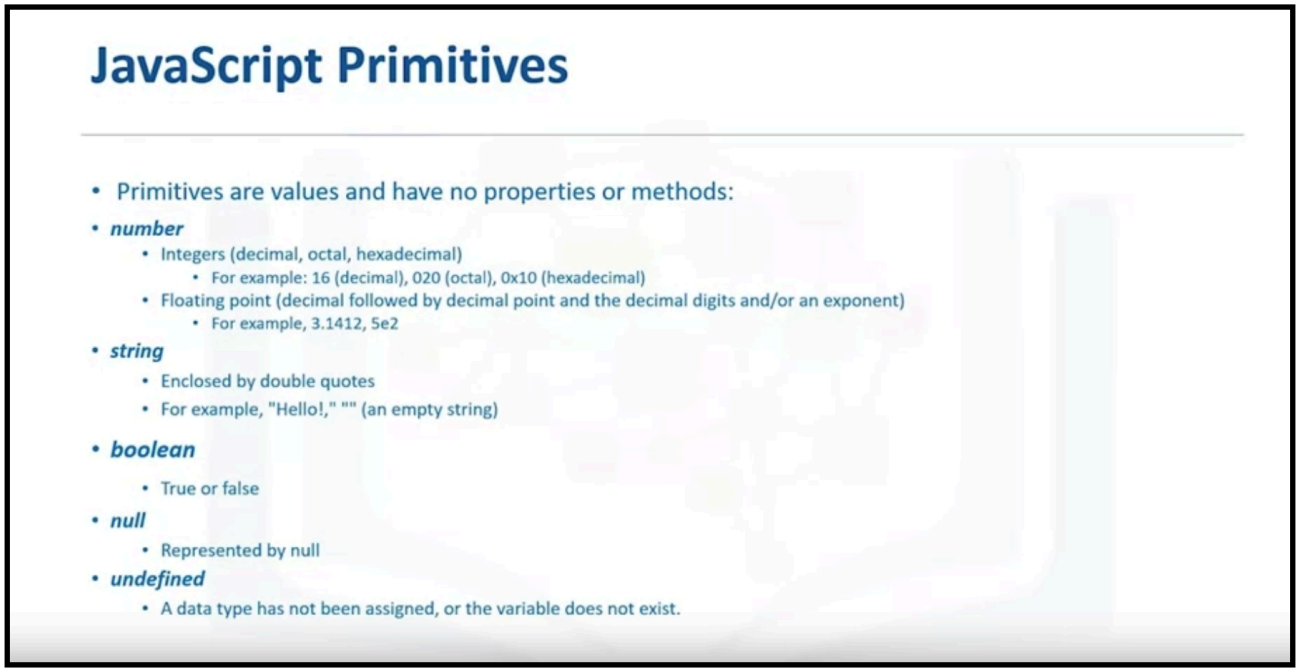
IBM FULL STACK DEVELOPMENT (Module 2)

1. Javascript





Wrapper Objects

- Some of the primitive data types have corresponding wrapper objects
- Allows an object of the related type to be created
- Stores a primitive value and offers methods with which to process it
- Wrapper objects have the same name as the primitive type, but they begin with a capital letter

PRIMITIVE TYPE	WRAPPER OBJECT
boolean	Boolean
number	Number
string	String

Array Objects

- Contain methods and properties used to store data
  - Accessible by indexed keys
  - Use a zero-based indexing scheme
  - Grow or shrink dynamically by adding or removing elements
  - Length property is the largest integer index in the array

Declaring Array Objects

- Declaration of an Array can use either an Array literal or an Array constructor
- Array constructors use the new Array keyword and specify the Array elements as parameters

```
numArray = new Array(0, 1, 2, 3, 4, 5);
numArray.length // returns 6
```

- Array literals create an array and initialize the elements in the Array

```
colors = ["red", "yellow", "green"];
```

Date Objects

- The Date object is a snapshot of an exact millisecond in time
  - Represented in the format YYYY-MM-DD 00:00:00 UT
- There are number of ways to provide parameters to the Date constructor
  - Example without a parameter:

```
var today = new Date(); // returns the local date and time
```

  - JavaScript automatically applies the toString() method if you attempt to display the date on a page or alert box

```
// Tues Jan 17 2012 13:15:00 GMT-8 (Pacific Standard Time)
```

Error Objects

- Error object instances are created when an exception is thrown
  - Properties of the error object instance contain information about the error
  - `message`: A description of the message
  - `name`: Identifies the type of error, such as `TypeError`, `RangeError`, or `URIError`

Custom Error Types

- Error object instances can be created by invoking one of the constructors for creating an error object
  - Custom error objects can be created
  - For example:

```
throw new Error("Only values 1 – 10 are permitted")
```

Declaration: var

- Traditional declaration method
- Has function-level scope

```
function example() {
    if (true) {
        var x = 10;
    }
    console.log(x); // 10 (available outside the if block)
}
```

Declaration: let

- Was introduced in ES6

Declaration: const

- Was introduced in ES6
- Declares constant values

```
const pi = 3.14;
// pi = 3.14159; // Error: Assignment to a constant variable

if (true) {
    const x = 10;
}
// console.log(x); // Error: x is not defined (outside the if block)
```

- Has block-level scoping

Naming convention

- Start: Letter, underscore (_), dollar sign ($)
- Contain: Letters, numbers, underscores (_), dollar signs ($)
- Case sensitive

```
let myVariable; // Valid variable name
let my_variable; // Valid variable name
let $price; // Valid variable name
let 2day; // Invalid variable name (starts with a number)
```

Primitive data types

String: Text

```
let name = "Alice";
```

Number: Integer and floating points

```
let age = 30;
let price = 9.99;
```

Boolean: True or false

```
let isStudent = true;
```

Primitive data types

Undefined: Declared but not assigned a value

```
Let x;
```

Null: Empty value

```
Let emptyValue = null;
```

Composite data types

Array: List-like data structure

```
my_array = [1,2,3,4,5]
```

Objects: Collection of key-value pairs

```
var person = {
name: "John",
age: 30,
city: "New York"
 {;
```

Recap

In this video, you learned that:

- JavaScript variables store and manage data with optional initialization using var, let, or const.
- var has function-level scope, let has block-level scope, and const is used for constants.
- Variable names are case-sensitive and start and contain a letter, underscore (_), or dollar sign ($).
- Primitive data types include strings, numbers, booleans, undefined, and null.
- Composite data types like arrays and objects handle multiple values and structured data.

Recap

In this video, you learned that:

- Control flow in JavaScript manages execution order using conditional statements.
- if statement executes code when a specified condition is true; otherwise, the code block is skipped.
- else if sequentially tests multiple conditions for situations with more than two possible outcomes, enabling various actions based on each condition.
- else specifies code to run when the if statement's condition is false, providing an alternative action.
- Nested if-else statements assess multiple conditions, executing distinct code blocks based on each condition's results.
- switch compares a value against multiple cases and runs code based on the first matching case, offering structured handling of multiple options.

Common APIs

The common APIs used in JavaScript applications are:

Document	Element	Window
document.getElementById(id)	element.innerHTML	window.open
document.getElementsByTagName(name)	element.style	window.onload
document.createElement(name)	element.setAttribute	window.scrollTo
parentNode.appendChild(node)	element.getAttribute	

Retrieving a node reference

- Use `document.getElementById` to:
  - Pass the id value as an argument
  - Return one specific HTML or XML element that is based on the id attribute in the node

Retrieving a node reference

- document.getElementsByTagName(tagName):
  - Retrieves a NodeList of elements with a specified tag name
  - Nodelist contains an array of elements in the document
  - tagName parameter is the literal name of the HTML tag

Example
If you run the function getElementsByTagName with a “p” as a parameter argument, a NodeList of all the paragraphs in the document is returned

Adding new nodes

`document.createElement(tagName)`

- Creates a new element with the namespace of the current document
- Functions can be used to place the element in the appropriate location

Example
insertBefore, appendChild, or replaceChild function can be used to add the newly created element into the document

Modifying an element's content

`element.innerHTML`

- Retrieves or sets the contents within an HTML element
- Returns all child elements as a text string
- Allows content change of an HTML element
- Removes all current child elements by setting value to string
- Browser parses the string and sets the contents of the HTML element

Modifying the inline style

- `element.style`
  - Retrieves or sets the inline CSS style for a particular element
  - Overrides setting from a CSS style sheet with one specific style

Set the style in JavaScript is with the forma:
`element.style.propertyName` = value

Modifying attributes

Methods	
element.setAttribute(attrName, attrValue)	• Dynamically modifies the attribute of an element • Example: document.getElementById("theImage").setAttribute("src", "another.gif");
element.removeAttribute(attrName)	Removes an attribute from an element
element.getAttribute(attrName)	Retrieves the value of the specified attribute in the element

Window object methods and events

`window.open(url, name, [features, replace])`

- Returns a reference to a new window object for the web browser

Parameters	
URL	Indicates the location of the web page to be displayed
Name	Specifies the name of the window
Features	Specifies the features of the window, such as its placement and dimensions
Replace	Optional Boolean value

Window object methods and events

Methods	
window.onload	Used to start a function after the page is loaded
window.dump("message")	Writes a string into the console for the web browser and is less intrusive
window.scrollTo(x-value, y-value)	Scrolls the web browser to a particular set of coordinates

Scripts that are tied to intrinsic events

- Scripts are executed every time a specific event occurs on a page
- Calling a function when an event occurs is called event binding

Event	Description
onload = script	Occurs when the user agent finishes loading a window
onclick = script	Occurs when the pointing device button is clicked over an element
onmouseover = script	Occurs when the pointing device is moved onto an element
onfocus = script	Occurs when an element receives focus
onkeyup = script	Occurs when a key is released over an element
onsubmit = script	Occurs when a form is submitted
onselect = script	Occurs when a user selects some text in a text field

DOM levels

Level 0
- Colloquially refers to DOM objects that existed before the DOM specifications were released

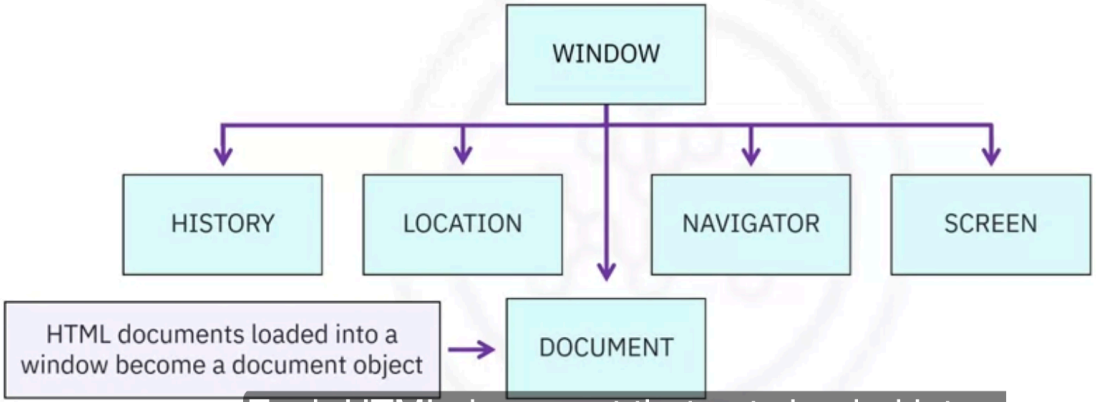
Level 1 (1998)
- Describes the low-level interface for structured documents (Core) and a specific version for the web (HTML)

Level 2 (2000)
- Includes a style sheet model and interfaces for manipulating the style information attached to a document
- In addition, an event model is included

Level 3 (2004)
- Specifies content models and document validation
- Includes Namespaces in XML, XML information Set, and XML schema

Level 4 (2015)
- Adds Mutation Observers as a replacement for Mutation Events

Basic DOM model for browsers



Each HTML document that gets loaded into a window becomes a document object.

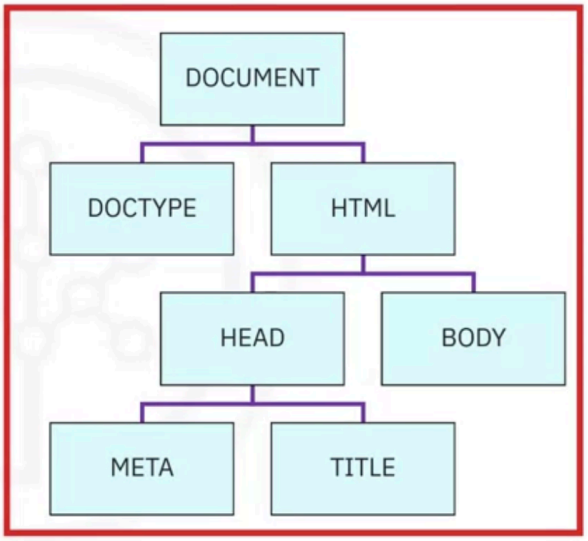
Client-side window object

- Window object dialog boxes:

WINDOW OBJECTS	DISPLAY	CODE
window.alert	Plain alert box	alert("message");
window.confirm	Confirmation OK/Cancel dialog	confirm("message");
window.prompt	Text-entry prompt	prompt("message", "defaultReply");

HTML document object diagram

```
<!DOCTYPE html>
<html>
    <head>
        <meta …>
        <title>Page Title</title>
    </head>
    <body>
    </body>
</html>
```



Recap

- DOM is the programming interface between HTML or XHTML and JavaScript
  - Each DOM levels provide a detailed set of features
  - Different browsers have different levels of compatibility with the DOM standard
- DOM for browsers is a hierarchy that includes objects, which perform different functions. For example, window, location, screen, and document objects
- DOM levels define object types that assists in building various documents

DOM HTML-related node types

DOM level 2 nodes that are applicable to HTML documents:

NODE TYPE (TEXT)	INTEGER VALUE	NODE NAME	NODE VALUE	DESCRIPTION
Element	1	Tag name	null	Any HTML tag
Attribute	2	Attribute name	Attribute value	A name-value pair
Text	3	#text	Text content	Text that is contained by the element
Comment	8	#comment	Text comment	HTML comment
Document	9	#document	null	document object
Document Type	10	DOCTYPE	null	DTD specification
Fragment	11	#document fragment	null	Nodes outside the document

DOM level 2 properties

DOM level 2 node object properties and corresponding data types:

OBJECT PROPERTY	DATA TYPE	DESCRIPTION
nodeName	String	Refer to previous table
nodeValue	String	Refer to previous table
nodeType	Integer	Integer constant – see previous table
parentNode	Object	Nearest containing object
childNodes	Array	All child nodes
firstChild	Object	First child
lastChild	Object	Last child
Attributes	NodeMap	Array of attributes

Recap

- How to list the related node types
- How to access nested objects using a dot notation
- How to name objects to make accessing them from the script easier