JavaScript

Tahaluf Training Center 2021

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

Day 2

1. JS Introduction
2. Document Object Model(DOM)
3. JS Output
4. JS Example
5. JS Popup Boxes
6. JS Variables

JavaScript, on the other hand, is a dynamic programming language that supports Math calculations, allows you to dynamically add HTML contents to the DOM, creates dynamic style declarations, fetches contents from another website, and lots more.

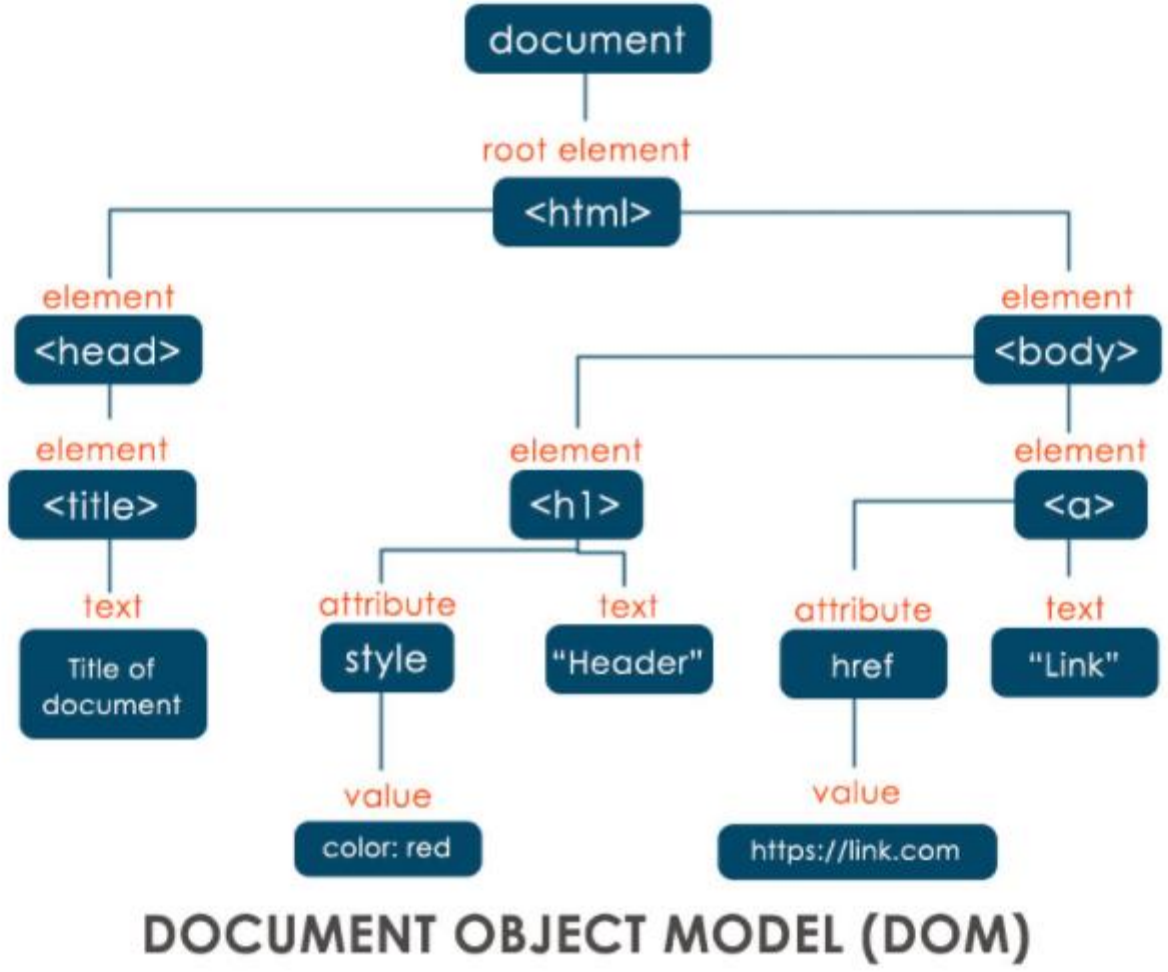
DOM is an acronym for **Document Object Model** which refers to a logical representation of a document (HTML or XML) and it's contents on the web. The representation is a tree structure showing the document in the browser from the root to all branches.

The DOM represents the document as nodes and objects. Each node in the tree is an object representing a part of the document.

Document Object Model

```
<html>
    <head>
        <title>Title of document</title>
    </head>
    <body>
        <h1 style='color: red'>
            Header
        </h1>
        <a href='https://link.com'>
            Link
        </a>
    </body>
</html>
```

Document Object Model



The DOM provides an API (Application Program Interface) called the **DOM API** which is used by programs and scripts like JavaScript to dynamically access and manipulate the document.

Manipulations include :

- Adding elements to HTML elements in a page.
- Remove elements from a page.
- Modify elements in a page.
  - Change attributes.
  - Change style of elements.
- Among others.

JavaScript can "display" data in different ways:

- Writing into an HTML element, using innerHTML.
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using console.log().

Example

```
function sum(a, b) {
    return a + b;
}
var total = sum(27, 10);
document.write(total);
```

Example

```
function sum(a, b) {
    return a + b;
}
var total = sum(27, 10);
console.log(total);
```

Example

```
function sum(a, b) {
    return a + b;
}
var total = sum(27, 10);
window.alert(total);
```

Alert Box

An alert box is often used if you want to make sure information comes through to the user. When an alert box pops up, the user will have to click "OK" to proceed.

```
alert('Hello Tahauf  Trainees');
```

Confirm Box

A confirm box is often used if you want the user to verify or accept something.
When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

```
confirm('Hello Tahauf  Trainees, Are you ready to start!');
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.
When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

```
prompt('Hello Tahauf  Trainees, Are you ready to start!(yes or no)!');
```

** Note: to display line breaks inside a popup box, use a back-slash followed by the character n

```
alert('Hello\n my name is Dana,\n Ready to start!');
```

JS Variables

JavaScript variables are containers for storing data values.

Declare variables with the var keyword.

```
var x = 5;
var y = 6;
```

ES2015 introduced two important new JavaScript keywords: let and const.

These two keywords provide **Block Scope** variables (and constants) in JavaScript.
Before ES2015, JavaScript had only two types of scope: **Global Scope** and **Function Scope**.

Global Scope

Variables declared Globally (outside any function) have Global Scope.
Global variables can be accessed from anywhere in a JavaScript program.

JS Variables


```
var trainerName = “Dana Kanaan";

// code here can use trainerName

function myFunction() {
  // code here can also use trainerName
}
```

Function Scope

Variables declared **Locally** (inside a function) have **Function Scope**.
Local variables can only be accessed from inside the function where they are declared.

JS Variables


```
// code here can NOT use trainerName

function myFunction() {
  var trainerName = “Dana Kanaan";
  // code here CAN use trainerName
}

// code here can NOT use trainerName
```

JavaScript Block Scope

Variables declared with the var keyword cannot have **Block Scope**.
Variables declared inside a block **{}** can be accessed from outside the block.

Before ES2015 JavaScript did not have **Block Scope**.
Variables declared with the let keyword can have Block Scope.
Variables declared inside a block **{}** cannot be accessed from outside the block:

JS Variables


```
{
  var x = 2;
}

// x CAN be used here
```

JS Variables

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

```
{
  let x = 2;
}

// x can NOT be used here
```

Redeclaring Variables

Redeclaring a variable using the var keyword can impose problems.
Redeclaring a variable inside a block will also redeclare the variable outside the block.

JS Variables


```
var x = 10;
// Here x is 10

{
  var x = 2;
  // Here x is 2
}

// Here x is 2
```

Redeclaring a variable using the let keyword can solve this problem.

Redeclaring a variable inside a block will not redeclare the variable outside the block

JS Variables


```
var x = 10;
// Here x is 10

{
  let x = 2;
  // Here x is 2
}

// Here x is 10
```

Loop Scope

Using var in a loop:

```
var i = 5;

for (var i = 0; i < 10; i++) {
  // some statements
}

// Here i is 10
```

Using let in a loop:

```
let i = 5;

for (let i = 0; i < 10; i++) {
  // some statements
}

// Here i is 5
```

In the first example, using var, the variable declared in the loop redeclares the variable outside the loop.

In the second example, using let, the variable declared in the loop **does not** redeclare the variable outside the loop.

When let is used to declare the i variable in a loop, the i variable will only be visible within the loop.

Block Scope

Declaring a variable with const is similar to let when it comes to **Block Scope**.

The x declared in the block, in this example, is not the same as the x declared outside the block.

JS Variables


```
var x = 10;
// Here x is 10

{
  const x = 2;
  // Here x is 2
}

// Here x is 10
```

Assigned when Declared

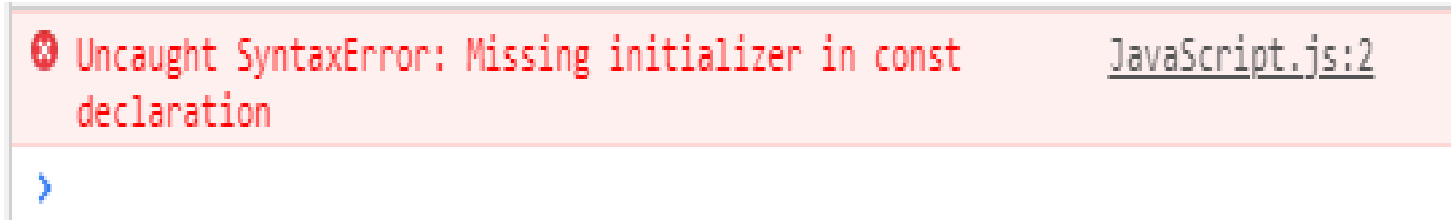
JavaScript const variables must be assigned a value when they are declared.

```
const PI = 3.14159265359;
```

JS Variables


```
const PI;
PI = 3.14159265359;
```

Incorrect



The keyword const is a little misleading.
It does NOT define a constant value. It defines a constant reference to a value.

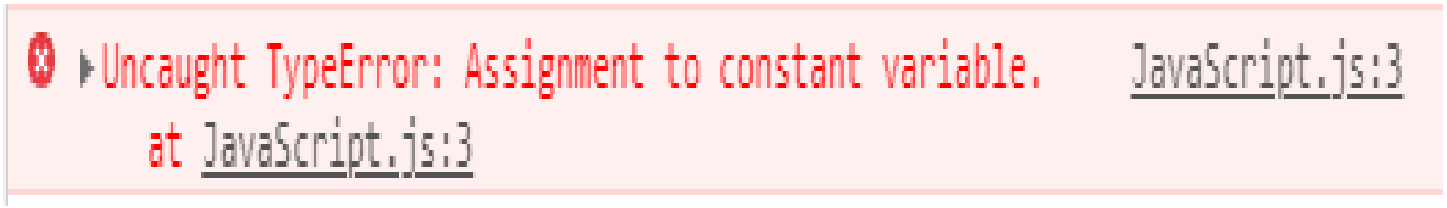
Because of this, we cannot change constant primitive values, but we can change the properties of constant objects.

JS Variables


```
const PI = 3.141592653589793;

PI = 3.14;      // This will give an error

PI = PI + 10;   // This will also give an error
```



Constant Objects can Change

You can change the properties of a constant object.

But you can NOT reassign a constant object.

JS Variables

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

```
// You can create a const object:
const car = {type:"Fiat", model:"500", color:"white"};

// You can change a property:
car.color = "red";

// You can add a property:
car.owner = “Dana";
```

JS Variables


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

car = {type:"Volvo", model:"EX60", color:"red"}; // ERROR
```

Constant Arrays can Change

You can change the elements of a constant array.

```
// You can create a constant array:
const cars = ["Saab", "Volvo", "BMW"];

// You can change an element:
cars[0] = "Toyota";

// You can add an element:
cars.push("Audi");
```

But you can NOT reassign a constant array.

```
const cars = ["Saab", "Volvo", "BMW"];

cars = ["Toyota", "Volvo", "Audi"];    // ERROR
```

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

On the E-Learning Portal