Javascript

A little history

The ancestry of JavaScript dates back to the mid 1990s, when version 1.0 was introduced for Netscape Navigator 2.

The European Computer Manufacturers Association (**ECMA**) became involved, defining **ECMAScript**, the great-granddaddy of the current language. At the same time, Microsoft introduced **jScript**, its own version of the language, for use in its Internet Explorer range of browsers.

The Browser Wars

In the late 1990s, Netscape Navigator 4 and Internet Explorer 4 both claimed to offer major improvements over earlier browser versions in terms of what could be achieved with JavaScript.

Unfortunately, the two sets of developers had gone in separate directions, each defining its own additions to the JavaScript language, and how it interacted with your web page.

This ludicrous situation forced developers to essentially write two versions of each of their scripts, and use some clumsy and often error-prone routines to try to determine which browser was being used to view the page, and subsequently switch to the most appropriate version of their JavaScript code.

Standardized DOM

Thankfully, the World Wide Web Consortium (the W3C) worked hard with the individual browser manufacturers to standardize the way web pages were constructed and manipulated, by means of the **Document Object Model (DOM).** Level 1 of the new standardized DOM was completed in late 1998, and Level 2 in late 2000.

Javascript comment

```
// This is a comment

/*  This comment can span

    multiple lines

    without needing

    to mark up every line  */
```

Primitive data type

1. Boolean -- true and false
2. Null -- null
3. Undefined -- undefined
4. Number -- 1,2,3, 1.2, 1e10, -1, 1e-10
5. String -- ‘name’ , “name”, ‘1’, ‘-1’
6. Symbol -- var a = Symbol() --> New in ES6/ECMAscript 2015

Number

Any value between -(2^53 -1) and 2^53 -1

the number type has three symbolic values: **+Infinity, -Infinity, and NaN**(not-a-number) (isNaN() to check if it is NaN).

To check for the largest available value or smallest available value within +/-Infinity, you can use the constants **Number.MAX_VALUE or Number.MIN_VALUE** and starting with **ECMAScript 2015(ES6)**, you are also able to check if a number is in the double-precision floating-point number range using **Number.isSafeInteger()** as well as **Number.MAX_SAFE_INTEGER and Number.MIN_SAFE_INTEGER**. Beyond this range, integers in JavaScript are not safe anymore and will be a double-precision floating point approximation of the value.

Null and Undefined

null: The value null represents the intentional absence of any value

undefined: A variable that has not been assigned a value is of type undefined

Variables and Dynamic type

var netPrice; // one javascript statement

netPrice = 8.99; // another javascript statement

netPrice = ‘hello’;

var productName = "Leather wallet";

alert(productName);

Also const and let ➔ new ES6

Operators

Plus + , subtract - , multiply * , divide /

Modulus division %

productCount++; // is same as below

productCount = productCount + 1;

productCount--; // is same as below

productCount = productCount – 1;

Note: 2.1 + 2.2 ?

Using the + Operator with Strings

```
var firstname = "John";
var surname = "Doe";
var fullname = firstname + " " + surname;
// the variable fullname now contains the value "John Doe"
```

Using functions

```
function sayHello() {

    alert("Hello");

    // ... more statements can go here ...

}

sayHello(); to call the function / to execute the function
```

PASSING ARGUMENTS TO FUNCTIONS

```
function cube(x) {
    alert(x * x * x);
}

function times(a, b) {
    alert(a * b);
}
times(3, 4); // alerts '12'

function cube(x) {
    return x * x * x;
}
var answer = cube(3);
```

Scope of the variables

1. Var in global is available any where
2. Var in function only available in side the function
3. Child function can use var that created in the parent function and parent function’s arguments

Object Basics

An object is a collection of related data and/or functionality (which usually consists of several variables and functions — which are called **properties** and **methods** when they are inside objects.)

```
var person = {
        [key]: value, // object is all about key and value pair.
        age: 32, // Properties
        greeting: function greeting() { // Methods
                alert('Hello!')
        }
};
```

Object Basics -- Dot and Bracket notation

To get value from Object

Person.age // 32 -> Dot notation

Person.name.first

Person.name.last

Person[‘age’] // 32 -> Bracket notation

Person[‘name’][‘first’]

To call method from Object

Person.greeting() // alert()

Person[‘greeting’]()

Set object properties and methods

Person.age = 45;
Person['name']['first'] = 'Cal';
Person.farewell = function farewell() { alert('good bye') }

The method and properties for string and number object String:https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/prototype

Number and Math:https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/prototype
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

‘this’ in object

-- ‘this’ is a keyword in javascript, which refers to the current object

```
Var persion1 = {
        Name: ‘chirs’,
        Greeting: function greeting() {
                console.log(this);
                alert(‘hi, i am’ + this.name + ‘.’);
        }
}
```

Here you can say, this is refering to persion1.

There will be several issues related to ‘this’ keyword, which will be covered later.

Array

```
var myArray = [];
var myArray = ['Monday', 'Tuesday', 'Wednesday'];
myArray.length  // returns 3 length is an unique property for array

var myArray = [];
myArray[0] = 'Monday';
myArray[1] = 'Tuesday';
myArray[2] = 'Wednesday';
```

Array is also an object

To manipulate array, here are some useful methods:

-- concat, join, toString, indexOf, lasIndexOf, slice, sort, splice, reverse, Array.isArray(), Array.from(), push, pop, unshift, shift, fill

map, filter, reduce, some, every, find, forEach

All method you can use for array
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

A little recap of data type and structure

- 6 primitive data type
- Array
- object

All above data can be stored in object or array

Var a = [{}, [], null, 1, ‘string’, undefined, Symbol()]
Var 2D-array = [[1, 1, 1], [2, 2, 2], [3, 3, 3]] // this is a 3 x 3 array
Var b = {
    A: [],
    B: {}
}
But be careful with keys in Object, only string or Symbol can be used as key for object, value could be any of them

typeof

typeof null => object

typeof undefined => undefined

typeof {} [] 1 ‘name’ function() {} true

DATA TYPE CONVERSION

var name = "David";
var age = 45;
name + age ?
1 + ‘1’ ? // any number + string = string
1 - ‘1’ ? // any number - string = number
‘1’ + ‘1’ ? //
‘1’ - ‘1’ ? => NaN
10 + '0' - 99 + '50' ? =>

Operator symbol to compare

OPERATOR SYMBOL	PURPOSE
==	Tests if LHS is equal to RHS
<	Tests if LHS is less than RHS
>	Tests if LHS is greater than RHS
<=	Tests if LHS is less than or equal to RHS
>=	Tests if LHS is greater than or equal to RHS
!=	Tests if LHS is not equal to RHS

=== & !== and diff with !== & !=

1. **== & !=** : This is a permissive value equality sign: it checks for equality between two variable after trying to parse them to the "best" format ("best" according the ECMA specifications)
2. === & !== : This is a less permissive value equality: it checks for value and type equality of the two evaluated variables. It is the recommended way to test for equality and is much more safer. Prefer it to the previous operator to avoid bad surprises.

Compare null and undefined

1. null === undefined
2. null == undefined == 0
3. null === null
4. null == null
5. !null / !0
6. isNaN(1 + null)
7. isNaN(1 + undefined)

Compare object

Var a = {}
Var b = {}

-- a === b ?

-- a == b ?

Shallow compare
Deep compare

Logical Operators

OPERATOR	SYMBOL
AND	&&
OR	||
NOT	!

The if else statement

Code:

```
If ( a > b ) { // if true
        console.log( ‘a is larger than b’ )
} else if (a == b) { // if false and a == b
        console.log( ‘a has the same value as b’ )
} else { // if false
        console.log( ‘a is smaller than b’ )
}
```

Multipue Conditions (else, else if)

```
var myAge = parseInt( prompt("Enter your age", 30), 10 );
if (myAge >= 0 && myAge <= 10) {
document.write("myAge is between 0 and 10<br />");
}
if ( !(myAge >= 0 && myAge <= 10) ) {
document.write("myAge is NOT between 0 and 10<br />");
}
if ( myAge >= 80 || myAge <= 10 ) {
document.write("myAge is 80 or above OR 10 or below<br/>");
}
if ( (myAge >= 30 && myAge <= 39) || (myAge >= 80 && myAge <=89) ) {
document.write("myAge is between 30 and 39 or myAge is " +
"between 80 and 89");
}
```

The switch Statement

Document Object Model

A **Document Object Model** (DOM) is a conceptual way of visualizing a document and its contents.

Each time your browser is asked to load and display a page, it needs to interpret (we usually use the word **“parse”) the source code** contained in the HTML file comprising the page.

As part of this parsing process, the browser creates **a type of internal model known as a DOM representation** based on the content of the loaded document. It is this model that the browser then refers to when rendering the visible page. You can use JavaScript to access and edit the various parts of the DOM representation, at the same time changing the way the user sees and interacts with the page in view.

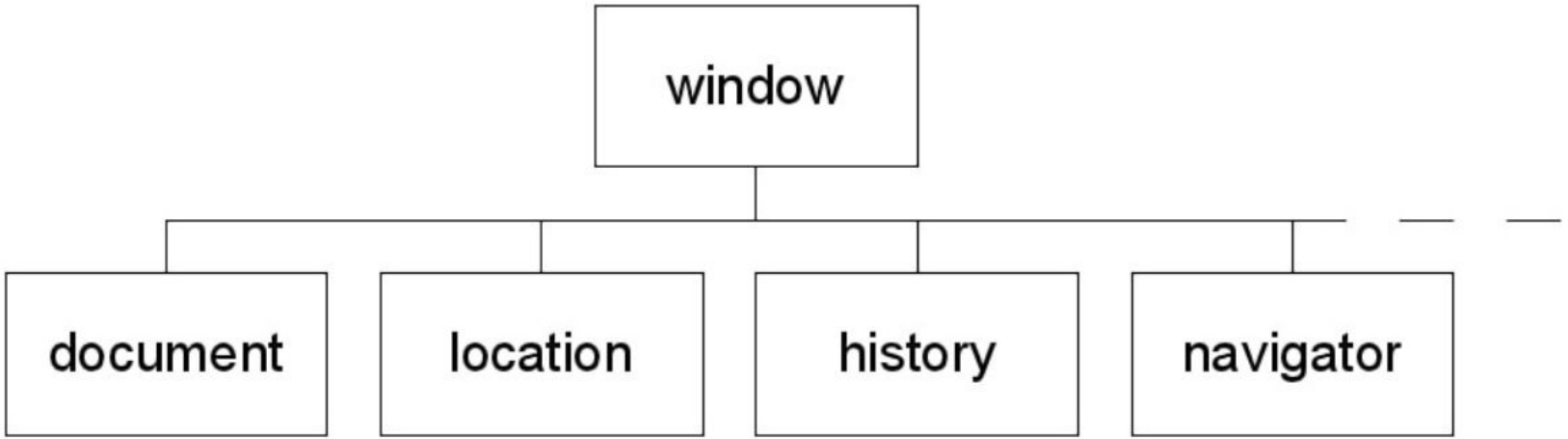
window and document Objects

Each time your browser loads and displays a page, **it creates in memory an internal representation of the page and all its elements**, the **DOM**. In the DOM, elements of your web page have a logical, hierarchical structure, **like a tree of interconnected parent and child objects (DOM Tree)**.

These objects, and their interconnections, form a conceptual model of the web page and the browser that contains and displays it. Each object also has a list of properties that describe it, and a number of methods **we can use to manipulate those properties using JavaScript(DOM manipulation).**

Right at the top of the hierarchical tree is the **browser window object**. This object is a parent or ancestor to everything else in the DOM representation of your page.

Window object and some of its children



1. the location object (containing details of the URL of the currently loaded page)
2. the history object (containing details of the browser's previously visited pages)
3. the navigator object (which stores details of the browser type, version, and capabilities)
4. The document object (containing every in your HTML)
5. Method to talk to the user
    a. window.alert("Here is my message")
    b. document.write("Here is another message")
6. Show msg in console: console.log(“Here is in the browser console”)

Access to the DOM by javascript

document.getElementById(‘elementId’)
document.getElementsByTagName(‘div’) => []
document.getElementsByClassName(name) => []
document.querySelector(‘#id.className[attribute=’id’]’) => single element
document.querySelectorAll(‘#id.className[attribute=’id’]’) => []

Above are all you need to access to DOM node in HTML page

DOM manipulation - I

var myDiv = document.getElementById("div1");
myDiv.innerHTML = this is div1;
myDiv.innerHTML = " Here is some new text instead! ";
myDiv.style.backgroundColor = ‘red’;
myDiv.style.color = ‘white’;
myDiv.getAttribute(‘data-id’);
myDiv.setAttribute(‘data-quantity’, ‘10’);
myDiv.classList.add(‘newClass’);

So every time you change anything to the DOM node is called DOM manipulation
https://developer.mozilla.org/en-US/docs/Web/API/Element/setAttribute

DOM manipulation - II

Element or DOM Node useful Api
=> About element/node control

parentNode -> node, parentElement -> node, children -> []
.contains() -> boolean,
.appendChild()
.removeChild()

DOM manipulation - III

Element or DOM Node useful Api => About attributes or class control

.setAttribute()
.removeAttribute()
.hasAttribute() ===>> diff from .hasAttributes()

.classList.toggle()
.classList.add()
.classList.remove()
.classList.replace()
.classList.contains()

DOM manipulation - IV

```
var newAnchor = document.createElement('a')
var container = document.getElementById('container')

newAnchor.innerText = 'This is new anchor'
newAnchor.setAttribute('href', 'https://www.google.com')
newAnchor.setAttribute('target', '_blank')


container.appendChild(newAnchor)
```

CAPTURING MOUSE EVENTS

<input type="button" onclick="alert('You clicked the button!')" value="Click Me" />

event.preventDefault()
event.currentTarget ===>> diff with event.target

Event Types

These event types belongs to the MouseEvent Object:

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

Useful Event Api Links

GlobalEventHandlers: (document.onload(DOM ready) vs body.onload(every resource is ready))
https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers

All Web Event => https://developer.mozilla.org/en-US/docs/Web/API/Event

Animation/Transition with Javascript:
https://css-tricks.com/controlling-css-animations-transitions-javascript/