JavaScript Basics

--- 1 ---

-> JS is a lightweight (doesn't consume much memory on your system), cross-platform (can be used on multiple platforms and systems), and OOP language.

-> JS is one of the three core technologies of web development.

-> Today, JS can be used in different places:

----> Client Side: JS was traditionally used only in the browser

----> Server Side: Thanks to node.js, can be used on server side

-> JS made the web development possible by providing dynamic effects and interactivity which helps build complex modern web applications.

-> Now there are various JS frameworks like Angular/React with different architectures that help build complex applications.

-> For developers to build apps using these F/Ws they need better understanding of JS.

ROLE OF JS IN WEB DEVELOPMENT

-> HTML, CSS and JS works together to create web pages

-> HTML is used to create content like buttons, tabs, etc.

-> CSS used for styling and for laying out the elements on the web page.

-> JS is the programming language that allows adding dynamic effects and real programming capabilities to the web pages.

Let's understand in terms of Noun, Adjectives and Verbs

HTML: Content: Nouns <p></p> means "paragraph"

CSS: Presentation: Adjectives p{color: red;} means "paragraph is red"

JS : Dynamic Effects/ Programming: Verbs p.hide() means "hide the paragraph"

JS Versions

ES5--> ES6/ES2015 -->ES7/ES2016 -->ES8/ES2017

--- 2 ---

VARIABLES AND DATA TYPES

-> Variable: is like a container where we place a value to be used over and over again in our code.

-> E.g. var Name='Radhika';

-> E.g. var age = 24;

-> Primitive JS Data Types: Number, String, Boolean, Undefined (Data Type of a variable that doesn't have a value yet), Null (Non -existent)

-> JS has a dynamic typing: data types are automatically assigned to variables. Needs to be careful what we do with our variables.

-> JS has this Camel Case kind on convention for writing names of variables.

-> E.g. var undef; //console: undefined

-> Variable Naming Conventions in JS:

---> Variable cannot start with numbers or symbols except $ or _ symbol

---> Cannot use reserved JS keywords as JS variable names

--- 3 ---

VARIABLE MUTATION AND COERCION

-> Comments: Single Line //

-> Comments: Multiline /*   */

-> E.g. var Name='Radhika';

-> E.g. var age = 24;

-> Type coercion

---> console.log (firstName + ' ' + age);

---> JS automatically converts int/bool/number to string to print on console, an example of type coercion

-> Variable mutation

---> E.g. age='twenty four'; // dont need to declare again, as we have already done that, and JS automatically type casts the variable (earlier int, now string)

-> Alert is a popup box which has an OK button - alternative to console

---> E.g. alert (firstName + ' is a ' + age + ' year old ' + job + '. Is he married? '+ isMarried);

-> Prompt popup box lets you enter some value and store it in a variable for future use

---> E.g. var lastName = prompt ('What is his last Name?');

--->   console.log (firstName + ' ' + lastName);

-- 4 --

-> BASIC OPERATORS

-> + * - / are Math operators in JS

---> e.g.   var now=2020;

--->   console.log (now + 2); //2022

--->   console.log (now * 2); //4040

--->   console.log (now / 10); //202

-> > < = are Logical operators in JS

---> e.g.   var johnOlder = ageJohn > ageMark;

--->   console.log (johnOlder); //true/false

-> typeof operator return the type of variable

---> e.g.   console.log (typeof johnOlder);//Boolean

--->   console.log (typeof ageJohn); //number

--->   console.log (typeof 'Mark is older than John'); //string

--->   var x;

--->   console.log (typeof x); //undefined

--- 5 --

-> OPERATOR PRECEDENCE

-> Multiple operators

-> JS executes math operators first and then logical operator is applied -- see the precedence table for more details

---> e.g.		var isFullAge = now - yearJohn >= fullAge; // true

--->		console.log (isFullAge);

-> Grouping

---> e.g. var ageJohn = now - yearJohn;

--->		var ageMark = 35;

--->		var average = (ageJohn + ageMark) / 2;

--->		console.log (average);

-> Multiple assignments

---> e.g. var x, y;

--->		x = y = (3 + 5) * 4 - 6; // 8 * 4 - 6 // 32 - 6 // 26

--->		console.log(x, y); //26 26

-> More operators

---> e.g. x *= 2;

--->		console.log(x); //52

--->		x += 10;

--->		console.log(x); //62

--->		x--;

--->		console.log(x); //61

-- 6 --

-> IF/ELSE STATEMENTS

-> Allows us to take decisions based on conditions : Control Structure

-> Syntax: if(condn1){----code----} else{----code----}

-> Syntax: if(condn1){----code----} else if(condn2){----code----} else{----code----}

-> e.g.

```
var firstName = 'John';
var civilStatus = 'single';

if (civilStatus === 'married') {
        console.log (firstName + ' is married!');
} else {
        console.log (firstName + ' will hopefully marry soon :)');
}
```

-- 7 --

-> BOOLEAN LOGIC

-> AND (&&) => True if ALL are TRUE

-> OR (||) => True if ONE is TRUE

-> NOT (!) => Inverts TRUE/FALSE value

-> AND and OR have lower precedence than logical operators

--- 8 --

-> THE TERNARY OPERATOR AND SWITCH STATEMENTS

->Syntax: condition? if TRUE : else THIS PART

```
-> e.g.   var age = 18;

                    var drink = age >= 18 ? 'beer' : 'juice';

                    console.log (drink); //beer
```

```
-> Syntax: switch (variable)

                         {

                         case 1 : {code}

                         case 2 : {code}

                         .

                         .

                         .

                         .

                         default : {code}

                         }
```

--- 9 --

-> TRUTHY AND FALSY VALUES AND EQUALITY OPERATORS

-> Falsy values: undefined, null, 0, '', NaN

-> Truthy values: NOT falsy values

->

----> Difference b/w == and ===

----> = is used for assigning values to a variable in JavaScript.

----> == is used for comparison between two variables irrespective of the datatype of variable. The == operator does type coercion!

----> === is used for comparision between two variables but this will check strict type, which means it will check datatype and compare two values.

--- 10 ---

-> FUNCTIONS

-> Functions are like containers that hold some lines of code and we can then pass arguments to it, which return a result.

-> Syntax: function <function name>(arguments){

}

var somename = <function name>(arguments); // calling a function

-> To avoid repeating same lines of code (i.e. DRY principle: DO NOT REPEAT YOURSELF) we use function.

--- 11 ---

-> FUNCTION STATEMENTS AND EXPRESSIONS

-> Another way to write function in JS (the above one still works)

-> Function declaration

```
function whatDoYouDo(job, firstName) {}
```

-> Function expression

```
var whatDoYouDo = function(job, firstName) {}

console.log(whatDoYouDo('teacher','john');
```

-> Anything that produces a immediate result, is a JS expression. E.g. 2+2=4, whatDoYouDo('teacher','abc') etc. // can be tried on browser console - F12

-> for loop, if-else statements, while loop etc that doesn't produce immediate result are JS Statements. e.g. if(true){console.log(10)}; //returns undefined - doesnt produce immediate result

-> function expression produce immediate result whereas function declaration do not.

-> for more details: Read FUNCTION DECLARATION VS FUNCTION Statements

https://link.medium.com/rjnyRRgrk5

-- 12 ---

-> ARRAYS

-> Initialize new array

```
var names = ['John', 'Mark', 'Jane'];

var years = new Array(1990, 1969, 1948);
```

-> Arrays are zero based, i.e. first element is 0th index.

-> To access elements of array :

```
console.log (names [2]); // Jane

console.log (names); // 3 'John', 'Mark', 'Jane'

console.log (names.length); // 3
```

-> Mutate Array Data

----> names [1] = 'Ben';

----> If we now add another element as names[5]='Mary' and then print the names array, we get ["John", "Ben", "Jane", empty × 2, "Mary"]

----> We can use names[names.length] = 'Mary'; to add at the last position of array.

-> Different Data Types

```
var john = ['John', 'Smith', 1990, 'designer', false];

john.push('blue'); // will push 'blue' to the end of array john

john.unshift('Mr.'); // will add Mr. as the first element

john.pop(); // remove element from the end

john.shift(); // remove element from the beginning --removes Mr.

john.indexOf(1990) //return the index of 1990 in the array

john.indexOf(23) // returns -1 as 23 is not in array
```

-- 13 --

-> OBJECTS AND PROPERTIES

-> In Objects we define key value pairs, which mean each value has a name i.e. key.

-> We declare objects using object literal, i.e. {}

```
            var john = {
                    firstName : 'John', //firstName is a property of john object
                    lastName : 'Smith',
                    year : 1990,
                    family : ['Jane','Bob','Mark','Emily'],
                    isMarried : false //last property will not have comma
            }; //semicolon at the end of object
```

-> Object can have different data types, arrays, and objects inside object (will use later)

->To access the properties of Objects:

```
            console.log (john); // here the protoype will be object
            console.log (john.firstName); //dot notation
            console.log (john['lastName']);//using key name
            var x = 'birthYear';
            console.log (john[x]); //assigning key to another variable
```

->To mutate the data:

```
            john.job = 'designer';
            john ['isMarried'] = true;
```

-> Another way to initialize object , using new Object();

```
            var jane = new Object();
            jane.firstName = 'Jane';
            jane.birthYear = 1969;
            jane['lastName'] = 'Smith';
            console.log (jane);
```

-> OBJECTS AND METHODS

-> We can have methods inside objects, like this

```
        var john = {
            firstName: 'John',
            lastName: 'Smith',
            birthYear: 1992,
            family: ['Jane', 'Mark', 'Bob', 'Emily'],
            job: 'teacher',
            isMarried: false,
            calcAge: function() {
                return 2020 - birthYear;
            } // this here is a function expression, which belongs to john object
        };
```

-> The function of object can be accessed by

```
    console.log (john.calcAge(1990)); --similar to the way we accessed other properties
```

-> Arrays have methods like pop(), shift() etc. which means array are objects because only objects can have methods. So, arrays are objects.

-> The year we passed above to call the method of john is already defined in the object, so to use that object property we can use JS keyword

'this'. So the function looks like this now:

```
   calcAge: function() {
      return 2020 - this.birthYear; // it means this is john.birthYear
   }
```

And we can access the object console.log(john.calcAge());

-> To store the value of this function in the object itself we can do something like this:

```
             calcAge: function() {
       this.age = 2020 - this.birthYear;
   }
```

We can call the method and log the object like this:

```
john.calcAge();
console.log (john); // will log age as a property of john object
```

-> An object has a special keyword 'this', which basically points to itself.

-- 15 ---

-> LOOPS - another control structure

-> We can automate repeating tasks through loops in JS

-> There are different types of loops in JavaScript

-> FOR LOOP

```
-> e.g. : for( var i = 0; i < 10 ; i++)
                              {
                                          console.log(i);
                              }
```

-> WHILE LOOP

```
-> e.g. : var i =0;
                  while(i<10)
                  {
                              console.log(i);
                              i++;
                  }
```

-> CONTINUE AND BREAK STATEMENTS

-> CONTINUE - To quit just the current iteration of the loop and then to continue to the next one

-> BREAK - Exits the current iteration and the entire loop and doest continue with the next iteration

-> NORMAL DIFFERENT OPERATOR !=

-> STRICT DIFFERENT OPERATOR !==

-> LOOPING BACKWARDS

```
-> e.g. : for (var i = john.length - 1; i >= 0; i--)
                    {
                            console.log(john[i]);
                    }
```

-- **16** --

-> JAVASCRIPT VERSIONS

-> 1996 : Changed from LiveScript to JavaScript to attract Java developers. JAVASCRIPT has almost nothing to do with Java.

-> 1997 : ES1 (ECMASCRIPT 1) became the first version of the JavaScript language standard:

ECMASCRIPT: The language standard;

JavaScript: The language in practice.

-> 2009 : ES5 (ECMASCRIPT 5) was released with lots of new features.

-> 2015 : ES6/ES2015 (ECMASCRIPT 2015) was released : the biggest update to the language ever!

-> 2015 : Changed to an annual release cycle.

-> 2016/2017/2018/2019 : Release of ES2016/ES2017/ES2018/ES2019...

-> WHICH VERSION TO USE:

-> ES5 : Fully supported in all browsers; Ready to be used today!

-> ES6/ES7/ES8 : Well supported in modern browsers; No support in older browsers; Can use most features in production with transpiling and polyfiling (converted to ES5).

-> ES9/ES10 : Future versions together called ESNext; Some features are supported in modern browsers;Can use most features in production with transpiling and polyfiling.

-> We are using ES5 for the first part of the project and ES6 in the later part.