JavaScript

Table of Content

What is JavaScript

- It is a verb of the web page that defines all the actions to be performed on a webpage
- Its an object oriented programming language that uses JIT compiler
- It is everywhere and all web browsers are installed with it.
- JS application ranges from web development, mobile development etc
- JS is easy, simple and very compatible with HTML-CSS
- It is must to have skill for any software engineer role

History

- JavaScript founder was Brendan Eich
- He developed JS in 1995
- He also developed first JS Engine, Spider Monkey which is still used by Mozilla firefox
- JavaScript name was later changed to 'mocha' and 'livescript' but it still remains JS due to some trademark reasons

Tools

For working with JavaScript, we just need 2 things

- Text editor
  - For writing and editing of JS code, we need a simple editor which can be notepad too.
  - But there are other powerful editors which provide additional functionalities like autocomplete, indentation, highlights etc.
  - Example: Visual Studio Code, Sublime text, Atom etc.
- Browser
  - All browsers come with an inbuilt JS engine.
  - They are used for executing JS code and also for debugging purposes.

JavaScript Hello World

- Before jumping right into coding, lets understand different ways to write js code
- JS basically gets merged into html which is a skeleton of web page
- We can write our JS in 3 ways
  - **Console**
    - Either just press Ctrl + Shift + I to open the console or you can right click and then go to inspect
    - Now since you are in console you can start writing your code

      ```
      console.log("hello world");
      ```
    - console.log is used to print output

- **Script tag**
  - **<script>** tag is used for writing javascript in HTML directly.
  - But every console.log() will print the output in console of browser

```
<!DOCTYPE html>
<html>
<head>
        <title>Home</title>
</head>
<body>
        <h1>My First JS Code</h1>
<script>
        console.log("Hello World!");
</script>
</body>
</html>
```

- **External file**
  - JS code is written in a separate file and is incorporated in the html using <script> tag.
  - This is the most efficient way.
  - It also enforces reusability and keeps code short, simple and easy to understand.

-------- html file ----------

```
<!DOCTYPE html>
<html>
<head>
        <title>Home</title>
</head>
<body>
        <h1>My First JS Code</h1>
<script type="text/javascript" src="home.js"></script>
</body>
</html>
```

-------- JS file ----------

```
console.log("Hello World!");
```

Comments

- Comments are extra info that helps in understanding of code and is not passed on to compilar at compilation.
- In JS, there are 2 types of comment
  - Single line

    //……..
  - Multi line

    /*

    …….

    …….

    */

JS Code

```
console.log("Hello World!");
/*console.log("!");
console.log("!");
console.log("!");
console.log("!");*/
//console.log("Hello !");
console.log("Bye");
```

Variables

- Variable is a name of a memory location where the data is stored.
- **Syntax** : var varname = val;
- In JS, variable is defined using var keyword
  - var name =”Faizan”;
  - var roll_no=27;

Naming Convention

- Variable names should begin with a letter, $ or an underscore(_).
- First character can be then followed by any combination of letters or digits.
- A variable name cannot be the same as any keyword as they are reserved for special purposes.
- Variable names case sensitive.
    - Eg: valid- apple, _name, $address1
    - Eg: invalid- 123apple, *roll, @email

Data Types

- JS is a dynamically typed language and does not need to specify data type explicitly of the variable.
- There are 5 primitive data types in JS
  - Number, String, Boolean, Undefined, Null
    - **Example**:

```
var num=23;
var string="hello";
var boolean=true;
var undef;
var nullValue=null;
```

num
23

string
"hello"

boolean
true

undef
undefined

nullValue
null

Type Coercion

- **Type coercion - when we are comparing 2 values of different types, the one type will force other to change it type as well so that comparison can be made possible**
  - **===** can stop coercion
  - if(1){} //1 will be converted to true
  - Object.is(param1,param2) is similar to === but it is different for some cases such as

    +0 === -0
    true
    Object.is(+0,-0)
    false

    NaN==NaN
    false
    Object.is(NaN,NaN)
    true

JS Properties

- **JS is dynamically typed- it doesn't has to specify the type of the variable**
  - There is no need to define the data type for variable like int, float etc.
  - Only let, var and const is used for variable declaration.

- **JS is weakly typed - type coercions is allowed in js**
  - when we are comparing 2 values of different types, the one type will force other to change it type as well so that comparison can be made possible
  - === can stop coercion

Unary Operators

- It needs one operand
- There are 2 major operators here i.e, increment and decrement and each one have 2 variations of postfix and prefix
- **Increment** - to increase the value by 1
  - postfix: first assign then increment
  - prefix: first increment then assign
  - Example

    var a=2; a++;

    2

    var a=2; ++a;

    3
- **Decrement** - to decrease the value by 1
  - postfix: first assign then decrement
  - prefix: first decrement then assign
  - Example

    var a=2; a--;

    2

    var a=2; --a;

    1

Shift Operators

- Two variations - left and right shift
- **left shift**: shift bits in left direction for specified value
- **right shift**: shift bits in right direction for specified value
- **Example:**

```
var  a=8,b=2;
console.log("a<<b : "+ (a<<b));
console.log("a>>b : "+ (a>>b));
```

Output

a<<b : 32

a>>b : 2

Relational Operators

- It comprises operators for comparisons
- There are operators to check inequality i.e., < ,>, <=, >=
- For equality check, we have 2 operators i.e., == and !=
- Difference between == and ===
  - == allow type coercion i.e, one type can change into another at the time of comparison
  - === does not allow type coercion
  - Note:
    - NaN==NaN
      false
    - NaN===NaN
      false
    - +0 == -0
      true
    - +0 === -0
      true

- **Example:**

```
console.log('2=="2" : '+ (2=="2"));
console.log('2==="2" : '+ (2==="2"));
console.log('2!="2" : '+ (2!="2"));
console.log('2!=="2" : '+ (2!=="2"));
console.log('2>"2" : '+ (2>"2"));
console.log('2>="2" : '+ (2>="2"));
console.log('2<"2" : '+ (2<"2"));
console.log('2<="2" : '+ (2<="2"));
```

Output

2=="2" : true
2==="2" : false
2!="2" : false
2!=="2" : true
2>"2" : false
2>="2" : true
2<"2" : false
2<="2" : true

Bitwise Operators

- It computes by going bit by bit
- Both side is checked irrespective of what first expression computes to
- 4 major bit operators are:
  - **&** - bitwise and, which returns 1 if both bits are 1 else 0.
  - **|** - bitwise or, which returns 1 if either of bits is 1 else 0.
  - **^** - bitwise xor, which returns 1 if both bits are different else 0.
  - **~** - bitwise not, which changes 1 to 0 and vice versa.
- **Example:**

```
var a=8,b=2;
console.log('a&b : '+ (a&b));
console.log('a|b : '+ (a|b));
console.log('a^b : '+ (a^b));
console.log('~a : '+ (~a));
```

Logical Operators

- are used for conditions comparison that basically checks for the validity of them
- They are also used in loops as part of termination conditions.
- If the first condition is enough to give the final verdict, it will not evaluate the second one.
- 3 operators are there:
  - **&&** - logical AND, returns true when both conditions evaluates to true
  - **||** - logical OR, returns true when either of the conditions evaluates to true
  - **!** - logical not, return true when condition evaluates to false
- **Example:**

```
var a=true,b=false;
console.log('a&&b : '+ (a&&b));
console.log('a||b : '+ (a||b));
console.log('!a : '+ (!a));
```

Output

a&&b : false

a||b : true

!a : false

Assignment & Ternary Operators

- **Assignment operator(=)** : is used to assign right hand side value to left hand side variable.
- **Ternary operator(?:)** : An alternative of if else. Condition is placed before ? and if evaluates to true then LHS of colon gets executed else RHS of colon will.
- **Example:**

```
var a=2;
console.log('a=2 : '+ (a=2));
console.log((a==2)?console.log("ok"):console.log("not ok"));
```

Output

a=2 : 2

ok

Conditional Statements - if else

- consist of 2 keywords, if and else
- This is a way where there are 2 paths possible depending upon a condition
- if condition manipulates to true then if gets executed otherwise code is else will execute
- you can multiple else if followed by else if there are more possible paths.
- Also nested if and else are possible
- **Example**:

```
var a=5,b=6;
if(a+b==11)
        console.log("equal");
else
        console.log("not equal");
```

Output

equal

Conditional Statements - switch

- It is an elegant way to replace multiple else-if
- **Syntax**:

```
switch(expression)
{
    case val: ... ;
              break;
    case val: ... ;
              break;
    case val: ... ;
              break;
    default: ...;
}
```

- Here depending upon the answer evaluated by the condition, case code gets executed.
- every case must be followed by break unless it is required not to as per logic. Otherwise, all cases will start executing from the matched case till either break is encountered or all cases gets exhausted
- default is optional and holds the code which should be executed if no catches gets matched.
- val can be integer, char or String in javascript.

- **Example**:

```
var day="sun";
switch(day)
        {
            case "mon":                 console.log("Today is monday");
break;
            case "tues":                console.log("Today is tuesday");
break;
            case "wed":                 console.log("Today is
wednesday"); break;
            case "thurs":       console.log("Today is thursday"); break;
            case "fri":                 console.log("Today is friday");
break;
            case "sat":                 console.log("Today is saturday");
break;
            case "sun":                 console.log("Today is sunday");
break;
            default :                   console.log("day is invalid!");
                    }
```

Loops - for

- It is best to use when we know the specified number of times the code should execute.
- **Syntax**:

  for(initialization; termination; updation){

  }

  - initialization - is used to initialize the variable, which is being used for iteration.
  - termination - comprises conditions which determine till when iteration will continue.
  - updation - how our variable will get updated.

- **Example**:

```
for(var i=0;i<5;i++)
    console.log("current value of i : "+i);
```

Output

current value of i : 0
current value of i : 1
current value of i : 2
current value of i : 3
current value of i : 4

Loops - while

- It is best to use when we know the specified expression depending on whose value the code should execute.
- **Syntax**:

  while(expression){

  }

  - expression- is used to dictate the condition who is responsible for loop continuation.

- **Example**:

```
var i=0;
while(i<5){
    console.log("current value of i : "+i);
    i++;
}
```

Output

current value of i : 0
current value of i : 1
current value of i : 2
current value of i : 3
current value of i : 4

Loops - do while

- It is best to use when we know that at least code must execute once irrespective of the condition.
- **Syntax**:

```
do{
}while( expression );
```

  - expression- is used to dictate the condition who is responsible for loop continuation.

- **Example**:

```
var i=0;
do{
    console.log("current value of i : "+i);
    i++;
}while(i<5);
```

Output

current value of i : 0
current value of i : 1
current value of i : 2
current value of i : 3
current value of i : 4

Arrays

- It is used to store ordered data together.
- It is defined within square brackets([ ]) and can have elements of different types
  var a=[1,true,'hello']
  a[2]
  "hello"
- You can also leave sime position
  var a=[2,3,,4]
  a[2]
  undefined
  a
  (4) [2, 3, empty, 4]

Arrays

- Array are special type of objects and new keyword can be used also to create array
  let arr = new Array(23,'cat',new Object());

  arr
  (3) [23, "cat", {…}]

  typeof(arr)
  "object"

Array Methods - Push and Pop

- They are used to add and delete elements from last respectively
- Example

```
let arr = [1,2];
arr.push(3);

arr
(3) [1, 2, 3]

arr.pop();
arr
(2) [1, 2]
```

Array Methods - Unshift and Shift

- They are used to add and delete elements from front respectively
- Example

```
let arr = [1,2];
arr.unshift(3);


arr
(3) [3, 1, 2]

arr.shift();
arr
(2) [1, 2]
```

Array Methods - Splice

- It used to add new elements in the array from specified to and from index
- Example

```
let arr = [1,2,3,4,5];
arr.splice(1,3,"hello");
arr
(3) [1, "hello", 5]
```

Elements from index 1 to 3(inclusive) are replaced by hello

- Splice can also be used to delete the elements from the array

```
let arr = [1,2,3,4,5];
arr.splice(1,3);
arr
(2) [1, 5]
```

Array Methods - Slice

- It can help one to create new array from existing array
- Example

```
let arr1 = arr.slice(1);
arr1
(4) [2, 3, 4, 5]

let arr = [1,2,3,4,5];
let arr1 = arr.slice(1,3);
arr1
(2) [2,3]
```

Elements from index 1 to 3(exclusive) are assigned to arr1

Array Printing

- Array can be printed using for, for each, for in or for of
- Using for

```
for(let i=0;i<arr.length;i++)
        console.log(arr[i]+" ");
```

23
cat
[object Object]

- Using forEach()

```
arr.forEach(item => console.log(item+" "));
```

Array Printing

- Using for of

```
for(item of arr)
        console.log(item+" ");
```

- Using for in
    - used for enumerables i.e, objects
    - it can also be used for iterables in which index acts as key

```
for(item in arr)
        console.log(item+" ");
```

0
1
2