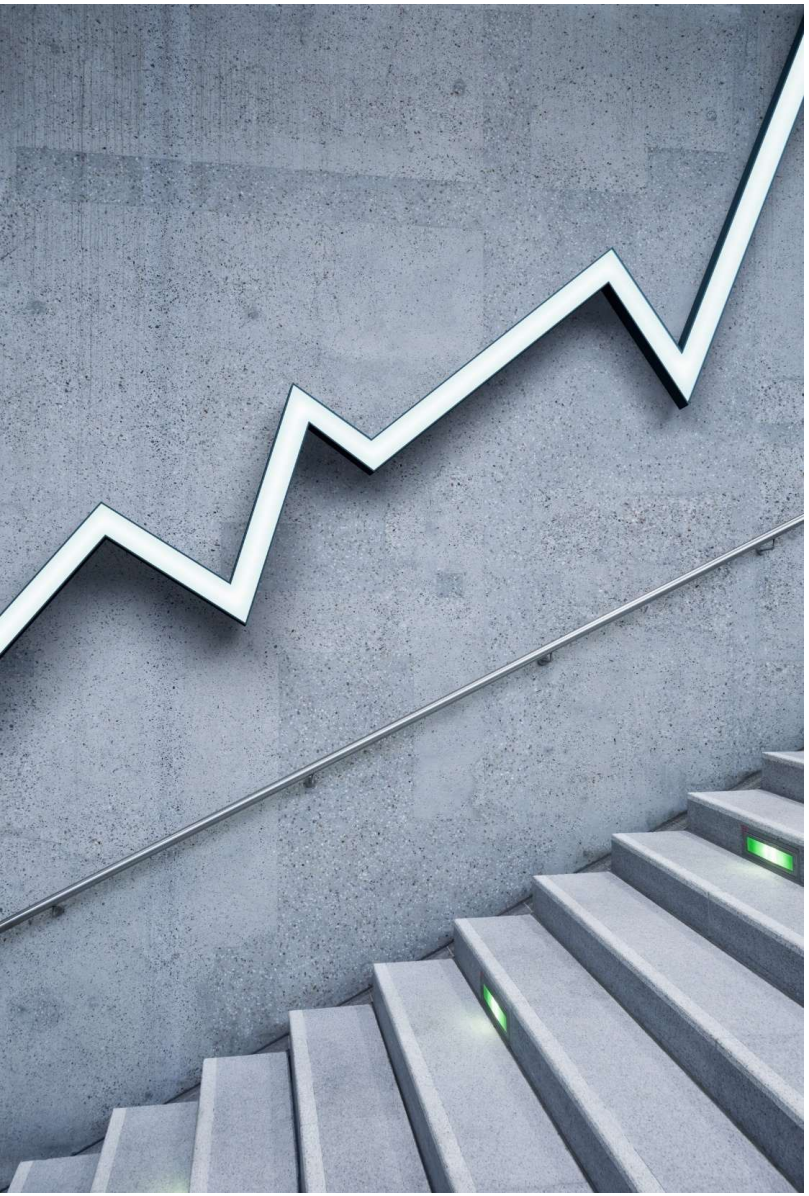

Learning JavaScript

PRAVEEN NAIR

What is JavaScript?

Used to program the behavior of web pages
JavaScript was invented by Brendan Eich in 1995.

JavaScript code is inserted between <script> and </script> tags.

Javascript was developed by Netscape

JavaScript vs VBScript (Microsoft)

Javascript supports all browser, vbscript supports IE

Originally Sun Microsystem and now Oracle

Basic Structure

```
let a=10;
let b=20;
let c = a + b;
console.log(c);
```

Variables

let

const (constant, can’t be changed)

Var –

var is function scoped and let is block scoped. Variable declared by let cannot be redeclared

Variables are case-sensitive, try camelCase, titlecase, with dash

Data Types (Primitive/Value type)

1. let n=2;
2. let s = “Hello World”; //double or single quote
3. let flag = true; //true or false - boolean
4. let name; //undefined
5. let cost=null;

Type conversion

```
let value = true;
 alert(typeof value); // Boolean
 value = String(value);

 let numStr="34";
 num = Number(numStr); // becomes a number 123

 alert(Boolean(num))

 /* Values that are intuitively “empty”, like 0, an empty string, null,
    undefined, and NaN, become false. Other values become true.*/
```

Comments // and /*

```
//let name='John';
    let age=20
  /*   document.write(name)
     console.log(name)
  */
```

Printing using backtick

```
let n=2;
let s = `Price of an apple is ${n}`;
console.log(s)
```

……..

Also called template literals….try multiline

Math Operators

Addition + (also concatenates string)

Subtraction -

Multiplication *

Division /

Remainder %

Exponentiation **

Comparison Operators

Operator	Description	Comparing	Returns
==	equal to	x == 8	FALSE
		x == 5	TRUE
		x == "5"	TRUE
===	equal value and equal type	x === 5	TRUE
		x === "5"	FALSE
!=	not equal	x != 8	TRUE
!==	not equal value or not equal type	x !== 5	FALSE
		x !== "5"	TRUE
		x !== 8	TRUE
>	greater than	x > 8	FALSE
<	less than	x < 8	TRUE
>=	greater than or equal to	x >= 8	FALSE
<=	less than or equal to	x <= 8	TRUE

Logical Operators

Logical NOT (!)

Logical AND (&&)

Logical OR (||)

Precedence of AND && is higher than OR ||

Assignments

A=4

A=3 + (b=4 + 6)

A=B=C=4+5 //chaining assignments

A++ // same as A=A+1

A- - // same as A=A-1

Conditional branching: if

```
let n = 7
if (n%2==0){
  console.log("Even Number")
}
else{
  console.log("Odd Number")
}
```

.....

Greater number
Greatest number
Vowel

Ternary/conditional operator ‘?’

```
let isEligible = (age > 18) ? true : false;
```

Try multiple condition

```
condition1
    ? true_expression1
    : condition2
        ? true_expression2
        : else_expression2
```

Nullish coalescing operator '??'

```
let count = 0;

let displayCount = count || 10;      // Output: 10 (because 0 is falsy)

let correctCount = count ?? 10;      // Output: 0 (because 0 is not null or undefined)
```

Switch statement

```
let price = 40;

switch (price) {
  case 30:
    alert( 'Too Cheap’ );
    break;

  case 40:
    alert( 'Perfect Price’ );
    break;

  case 50:
    alert( 'Too Costly’ );
    break;

  default:
    alert( "I don't know the price" );

}
```

while loop

```
while (condition) {
  ...
}
```

For loop

```
for (let i = 0; i < 3; i++) {
  alert(i);
}
```

Try break and continue

JavaScript Regular Function

```
function showMsg() {
  alert( ‘Hello World!' );
}

showMsg();
```

(IIFE)immediately invoked function expression

```
(function functionName() {
  console.log("Hello World");
})();
```

Passing arguments

```
function sum(a, b) {
  c = a + b;
 alert(c);
}
sum(1, 2);
```

Returning Values

```
function sum(a, b) {
  return a + b;
}
let result = sum(1, 2);
alert( result ); // 3
```

Function Expressions

```
let sayHello = function() {
        alert( "Hello World" );
    };

sayHello();
```

Arrow functions

```
let result = (a, b) => {
            let c = a + b
            return c
    };


let result = function(a, b) {
      let c = a + b
      return c;

  };


alert(result(3, 2) );
```

Callback functions

```
function ask(question, yes, no) {
  if (confirm(question))  yes()
  else no();
}
function a() {
  alert( "You agreed." );
}
function b() {
  alert( "You canceled." );
}
msg = “Do you agree?”
ask(msg, a, b);
```

Functions (...args) vs arguments

```
function sum(){
    let sum=0
    for (let i=0;i<arguments.length;i++){
      sum = sum + arguments[i]
    }
    alert(sum)
  }
  sum(2,3,4,5)
```

.....

```
function sum(...args) {
    let sum = 0;
    for (let i = 0; i < args.length; i++) {
      sum = sum + args[i];
    }
    console.log(sum);
  }
  sum(2, 3, 4, 5);
```

Data Types (Reference Type)

1. Objects
2. Arrays
3. Functions

Objects – Keyed Collections

```
let student = {
  name: "Smitha",
  age: 30
};
console.log(student.name) // student[“house address”]
console.log(student.age)
Console.log(student)
```

Objects – add / delete properties

Student.iseligible=true

Delete student.iseligible

Console.log(student)

```
const arr = Object.entries(student);

const keyArr = Object.keys(student);

const valueArr = Object.values(student);

console.log(arr,keyArr,valueArr)
```

Objects – lookup

```
marks = {
  “John”:30,
  “Joe”:60
}

name = “John”
Console.log(Marks[name])
```

Arrays

```
let arr = ["Mango", "Orange", "Cherry"];
for (let i = 0; i < arr.length; i++) {
  console.log( arr[i] );
}
```

.....

Array Methods – foreach, map

```
let fruits = ["apple", "mango", "orange"];
    fruits.forEach((value,index,arr) => {
      console.log(value,index,arr);
    });
let fruits = ["apple", "mango", "orange"];
    fruits.map((value, index, arr) => {
      console.log(value, index, arr);
    });
```

Array Methods – filter and find

```
let score = [34, 12, 67, 89, 30];
    let result = score.filter((v) => {
      return v > 40;
    });
    console.log(result);
```

```
let empnum = [1003, 1005, 1006, 1034];
    let result = empnum.find((v) => {
      return v == 1003;
    });
    console.log(result);
```

Arrays – reduce method

```
let marks = [40,60,80,40]
    let sum = marks.reduce((total,value)=>{
        return total + value
    })
    console.log(sum)
```

Module Import/Export

```
calc.mjs
function add(x,y){
    return x+y
}
export default add
.................................
import add from "./calc.mjs"

        let sum = add(4,5)

        console.log(sum)
```

Module Import/Export - multiple

```
calc.mjs
function add(x,y){
    return x+y
}
function subtract(x,y){
    return x-y
}

export {add, subtract}
.................................
 import {add,subtract} from "./calc.js“
        let sum = add(4,5)
        console.log(sum)
        let difference = subtract(8,3)
        console.log(difference)
```

Var vs let keyword

```
var a = 20 //function scope
 if (10>4) {
      var a=10
  }
console.log(a)
.........................................
 let b = 20 //function score + block scope
if (10>4) {
   let b=10
}
console.log(b)
```

Spread Operator (...)

```
let marks = {};
marks = {...marks,English:95}
console.log(marks)
marks = {...marks,Maths:90}
console.log(marks)

let students = ["John","Cathy","Mike"]
students = [...students,"Amy"]
console.log(students)
students = [...students,"Alice"]
console.log(students)
```

Error Handling – reference error

```
try{
        console.log(a)
    }
 catch(err){
        console.log(err)
        console.log(err.message)
        console.log(err.name)
    }
```

JavaScript Object Notation (JSON)

```
{
  "name": "Alice",
  "age": 25,
  "isStudent": false,
  "skills": ["HTML", "CSS", "JavaScript"],
  "address": {
    "city": "Chennai",
    "pincode": 600001
  }
}
```

JSON.stringify

```
student = {name:"john",age:20,pass:true}
student = JSON.stringify(student)
console.log(student)
```

expected output:

```
{"name":"john","age":20,"pass":true}
```

JSON.parse

```
let student = '{"name":"john", "age":20}'
console.log(student.name)
let obj = JSON.parse(student)
console.log(obj.name)
```

Why promise is needed

```
//asynchronous : occurring at the same time
    const f1 = () => {
      setTimeout(() => {
        return 5;
      }, 5000);
    };

    const f2 = (x) => {
      console.log(x + 6);
    };

    let n1 = f1();
    f2(n1);
```

Use callback to solve the issue

```
const f1 = (fnc) => {
      setTimeout(() => {
        fnc(5);
      }, 5000);
    };

    const f2 = (x) => {
      console.log(x + 6);
    };

    f1(f2);
```

Use promise and .then

```
const f1 = () => {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        resolve(5); //use resolve instead of return
      }, 5000);
    });
  };

  const f2 = (x) => {
    console.log(x + 6)
  };

  f1().then((a) => f2(a));
```

Async/await

```
const f1 = () => {
      return new Promise((resolve, reject) => {
        //   resolve(5);
        reject("Something went wrong");
      });
    };
    const f2 = () => {
      console.log("Function 2");
    };
    const f3 = async () => {
      try {
        let n1 = await f1();
        f2(n1);
      } catch (err) {
        console.log(err);
      }
    };
f3()
```

Fetch API

```
fetch("https://jsonplaceholder.typicode.com/users")
  .then((res) => res.json())
  .then((data) => {
    data.map((value) => {
      console.log(value.name);
    });
  });
```

Closure (access to outer variable)

```
function main() {
    let b = 1;
    function sub() {
      return b;
    }
    return sub;
  }
  let f1 = main();
  console.log(f1());
  console.log(f1());
```

Target Event

```
<p onclick="alert(event.target.innerHTML)">This is a dummy text</p>

<input type="text" onchange="alert(event.target.value)" /><br /><br /><br />
```

Accessing element's value

```
<p id="p2">This is a paragraph</p>
  <input type="text" id="t1" placeholder="Enter a value">    <br>
  <input type="button" onclick="myfunc()" value="Submit">
  <script>
   function myfunc(){
    alert(document.getElementById('t1').value)
   }

  </script>
```

Thank You

- PRAVEEN NAIR