Javascript

Overview

- For effective and efficient web development, developers must know three languages:
  - HTML – for creating the web page and defining its content
  - CSS – for specifying the layout and styles of the various elements
  - Javascript – for programming the behaviour of the web page

- JS is an easy to learn, lightweight, interpreted scripting language.
- It was created by a Netscape developer named Brendan Eich in 1995.
- It can run on both client side as well as server side web browsers.

Four Important Questions

1
What is JavaScript?

- One of the **most popular and widely used programming language**.
- **Capable of →** Front-end, Back-end, Full Stack development

2
What can JS do?

- Build Web/ Mobile applications, Real-time networking apps like chats, video streaming services, Games, etc.

3
Where does JS code run?

- **JS engine present in a browser** (SpiderMonkey, V8) **or outside a browser** (Node) presents the run-time environment for JS code

4
JavaScript vs ECMAScript?

- **ECMAScript** - specification that provides the rules, details, and guidelines that a scripting language must observe to be considered ECMAScript compliant.
- **JavaScript** – programming language that conforms to ECMAScript

JS Basics

Inclusion, Output, Variables, Data Types, Operators

JavaScript Inclusion

- JavaScript code is inserted between <script> and </script> tags.
- Any number of scripts can be placed in a web page.
- Three ways of placing and using a script:

  Inside <head> section

  Inside <body> section

  External JS file

JavaScript Output

- JS can "display" o/p in different ways, by using

  innerHTML - writes into an HTML element

  document.write() - writes into the HTML output

  window.alert() / alert() – writes into a pop up alert box

  console.log() – writes into the browser console

JavaScript Variables

- Variables are identified using unique names called *identifiers*.
- General rules for naming variables are:
  - Can contain letters, digits, underscores, and dollar signs
  - Must begin with a letter, $ and _
  - Are case sensitive (x and X are different variables)
  - Keywords cannot be used as names
  - Must not contain any white-space, or special character (!, @, #, %, ^, &, *)
- *Variable Scope – Local, Global, Block* (`var` | let | const)

Declaring & Defining Variables

```
var firstNumber;
firstNumber = 10;
```
$\rightarrow$ Declaring a variable

$\rightarrow$ Assigning value to declared variable

```
const x = 10;
```
$\rightarrow$ Declaring + Assigning value

```
let fname, age;
fname = "Anu";
age = 12;
```
$\rightarrow$ Declaring multiple variables in one statement and then assigning values to them

```
let fname = "Anu", age = 12, city = "Noida";
```
$\rightarrow$ Declaring and assigning values to multiple variables in one statement

JavaScript Data Types

Primitive/ Value Data Types

- String
- Number
- Boolean
- Undefined
- Null

Non-Primitive/ Reference Data Types

- Object
- Array

Primitive Data Types

```
let name = ‘Anu’;  // string literal

let age = 12;  // number literal

let flag = true;  // boolean literal

let grade = undefined;  // undefined literal

let color = null;  // null literal
```

Non-Primitive Data Types

```
// Object
let student = {
    name: 'Anu',
    age: 12
};

//Dot Notation
student.name = 'Avi';

//Bracket Notation
student['name'] = 'Mini'

console.log(student.name);
```

```
// Array
let colors = ['red', 'blue'];

console.log(colors);

console.log(colors[1]);

colors[2] = 'yellow';

console.log(colors);

console.log(colors.length);
```

JavaScript Operators

JavaScript supports the following types of operators:

- Arithmetic [+, -, *, /, **, %, ++, --]
- Assignment [=, +=, -=, *=, /=, %=, **=]
- Comparison [==, ===, !=, !===, >, <, >=, <=]
- Logical [&&, ||, !]
- Bitwise [&, |, ~, ^, <<, >>, >>>]
- Miscellaneous [*ternary* (? :) and *typeof*]

JS Functions

JavaScript Functions

Function is a reusable block of code, which on invocation performs a particular task.

```
function name(parameter1, parameter2, parameter3)
{
  // statements or code to be executed
}
```

- **Syntax** – *function* keyword, followed by a name, followed by parentheses
- **Parameters** - multiple comma separated parameters, can be passed to a function
- **Definition** – function has to be defined before use
- **Invocation** – function executes only when invoked

JavaScript Functions

- Functions as *Variable*
- Function *Invocation*
- *Parameters* and *Arguments*
- *Return* Statement

```
const c = 5;

function f1(a, b)
{
    let f = a + b + c;
    return f;
}

console.log('Sum: ' + f1(1,2));
```

User-defined | Arrow | Built-in Functions

```
const c = 5;

function f1(a, b)    // User-defined Function
{
    let f = a + b + c;
    return f;
}

let f2 = (a, b) => a * b;    // Arrow Function

console.log('Sum: ' + f1(1,2)); // Built-in Function
console.log('Product: ' + f2(1,2)); // Built-in Function
```

FUNCTION DECLARATION

FUNCTION EXPRESSION

Arrow Function

```
let sum = (a, b) => a + b;

/* This arrow function is a shorter form of:

    let sum = function(a, b) {
        return a + b;
    };
*/

alert(sum(1, 2));
```

Arrow Function

- If there is only one parameter being passed, then parentheses around parameters can be omitted in arrow function.

```
const b = 10;
let sum = a => a + b;

alert(sum(2));
```

- If there are no parameters, then empty parentheses have to be placed (parentheses cannot be omitted in this case).

```
sayHello = () => alert("Hello");

alert(sayHello());
```

JS Objects

JavaScript Objects

- JS object is an entity having *state* (properties) and *behavior* (method)
- Ways to create objects:

Object literal

```
// Object Literal
let emp1 = {
    id:101,
    name:'Sameer Gupta'

    printDetails: function() {
        console.log(this.id+" "+this.name);
    }
}
emp1.printDetails();
```

JavaScript Objects

- JS object is an entity having *state* (properties) and *behavior* (method)
- Ways to create objects:

Object literal

Object instance

```
//Object instance
let emp = new Object();
emp.id = 101;
emp.name = 'Sameer Gupta';
console.log(emp.id+" "+emp.name);
```

JavaScript Objects

- JS object is an entity having *state* (properties) and *behavior* (method)
- Ways to create objects:

Object literal

Object instance

Factory function

```
// Factory Function
function createEmp(id,name) {
    return {
        id,
        name,

        printDetails: function() {
            console.log(this.id+" "+this.name);
        }
    }
}

let x = createEmp(101,'Sameer Gupta');
x.printDetails();
```

JavaScript Objects

- JS object is an entity having *state* (properties) and *behavior* (method)
- Ways to create objects:

Object literal

Object instance

Factory function

Constructor function

```
//Constructor
function Emp(id,name) {
    this.id = id;
    this.name = name;

    this.printDetails= function() {
        console.log(this.id+" "+this.name);
    }
}

let x1 = new Emp(101,'Sameer Gupta');
x1.printDetails();
```

JavaScript Classes

- A class is an *extensible program-code-template for creating objects*, providing *initial values for state* (*member variables*) and *implementations of behavior* (*member functions or methods*). [1]

- Useful for creating many objects of the same kind.

```
class ClassName {
    constructor() { ... } // automatically called by new; initializes the object;
                             constructor is a keyword;
    method1() { ... }
    method2() { ... }
    method3() { ... }
    ...
}
```

1 https://en.wikipedia.org/wiki/Class_(computer_programming)

JavaScript Classes

```
// Class
class Emp {
  constructor(name, id) {
    this.name = name;
    this.id = id;
  }

  prnDetails() {
    console.log(this.id + " " + this.name);
  }

}

// Usage:
let emp1 = new Emp("Sameer", 101);
emp1.prnDetails();

let emp2 = new Emp("Richa", 102);
emp2.prnDetails();
```

Object Destructuring

Object destructuring provides an alternative way to assign properties of an object to variables.

```
let student = {
    name: 'Sameer',
    rollno: '101',
    branch:'CSE'
};

// using dot notation
let s_name = student.name;
let s_rollno = student.rollno;
let s_branch = student.branch;

// Object Destructuring
let {s_name,s_rollno,s_branch} = student;   //if you want to use all variables

let {s_name} = student; //if you want to use only one variable

let {rollno: rn} = student //assign an alias to the variable name as per your choice
```

Spread Operator

- A new addition to the set of operators in JavaScript ES6
- It takes an iterable (*like an array*) and expands it into individual values
- Denoted by three dots (…)
- Some uses of spread operator:

  Copy an array

  Concatenating arrays

  Passing array as argument

  Combining objects

Uses of Spread Operator

```
//Copying an array
let arr1 = [1,2,3];
let arr2 = [...arr1];
console.log(arr2);
```

```
//Concatenating arrays
let arr11 = [1,2,3];
let arr12 = [4,5,6];
let arr13 = [...arr11, ...arr12, 7,8,9]
console.log(arr13);
```

```
//Array as arguments
function add_num(n1, n2, n3) {
  console.log(n1 + n2 + n3);
}

let numbers = [1,2,3];
add_num(...numbers);
```

```
//Combining Objects
let obj1 = { name:'Sameer' };
let obj2 = { rollno: 101 };

let obj3 = {...obj1, ...obj2, branch:'CSE'};

console.log(obj3);
```

Useful Resources

- https://www.w3schools.com/js/default.asp
- https://www.tutorialspoint.com/javascript/index.htm
- https://data-flair.training/blogs/javascript-tutorial/
- https://www.javascripttutorial.net/
- https://javascript.info/