Javascript

**1.What is Javascript.**

* JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages.
* It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.
* It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser. Since then, it has been adopted by all other graphical web browsers.
* With JavaScript, users can build modern web applications to interact directly without reloading the page every time.
* The traditional website uses js to provide several forms of interactivity and simplicity.
* Javascript is used for image manipulation, form validation,and dynamic changes in contents.

**2.What are the features of Javascript.**

* All popular web browsers support JavaScript as they provide built-in execution environments.
* JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
* JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
* It is a light-weighted and interpreted language.
* It is a case-sensitive language.
* JavaScript is supportable in several operating systems including, Windows, macOS, etc.
* It provides good control to the users over the web browsers.

**3.What are use of Javascript.**

* JavaScript is used to create interactive websites.
* It is mainly used for:
  + Client-side validation, Dynamic drop-down menus,
  + Displaying date and time,
  + Displaying clocks.
  + Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),

**4.How we put Javascript code inside a document.**

* JavaScript provides 3 places to put the JavaScript code: within body tag, within head tag and external JavaScript file.

**<script** type="text/javascript"**>**  alert("Hello Javatpoint");  **</script> //body**

**5.Explain Advantages External Javascript file.**

* We can create external JavaScript file and embed it in many html page. It provides code re usability because single JavaScript file can be used in several html pages.
* An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.
* It allows easy code readability.
* It enables both web designers and coders to work with html and js files parallelly and separately, i.e., without facing any code conflictions.
* The length of the code reduces as only we need to specify the location of the js file.

**6.Explain Disadvantages External Javascript file.**

* The stealer may download the coder's code using the url of the js file.
* The web browser needs to make an additional http request to get the js code.
* If two js files are dependent on one another, then a failure in one file may affect the execution of the other dependent file.

**7.Explain JS Comments.**

* It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.
* The JavaScript comment is ignored by the JavaScript engine.
* It is represented by double forward slashes (//).
* It used to make code easy to understand.
* It used to avoid the unnecessary code.

**8.Explain JS Variable.**

* Variables are containers for storing data (storing data values).
* There are two types of variables in JavaScript : local variable and global variable.
* A JavaScript local variable is declared inside block or function. It is accessible within the function or block only.
* A JavaScript global variable is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable.

**9.Explain rules for declaring JS Variable(also known as identifiers).**

* Name must start with a letter (a to z or A to Z), underscore( \_ ), or dollar( $ ) sign.
* After first letter we can use digits (0 to 9), for example value1.
* JavaScript variables are case sensitive, for example x and X are different variables.

**10.Explain JS Datatypes.**

* JavaScript provides different data types to hold different types of values. There are two types of data types in JavaScript.
* Primitive data type:

A datatype has a value that has no properties or methods is called as primitive datatype

Primitive datatype values are immutable (they are hardcoded and cannot be changed). if x = 3.14, you can change the value of x, but you cannot change the value of 3.14.

Ex: String, Number, Boolean, Undefined, Null, Symbol, Bigint

Non-primitive (reference) data type: Object, Array, RegExp

**11.Explain JS Arithmetic Operators.**

* Arithmetic operators are used to perform arithmetic operations on the operands.
* Addition(+), Subtraction(-), Multiplication(\*), Division(/),

Modulus/Reminder(%), Increment(++), Decrement(--)

**12.Explain JS Comparison Operators.**

* Comparison operators are used to compare the two operands.
* Equal to(==), Equal value and equal type(===), Not equal (!=),

Not Equal value and equal type(!==),

Greater than(>), Less than(<), Less than or equal(<=)

**13.Explain JS Bitwise Operators.**

* The bitwise operators perform bitwise operations on operands.
* AND(&), OR(|), NOT(~)

**14.Explain JS Logical Operators.**

* AND(&&), OR(||), NOT(!)

**15.Explain JS Assignment Operators.**

* Assign(=), Add and Assign(+=), Subtract and Assign(-+),

Multiply and Assign(\*=), Divide and Assign(/=)

Modulus and Assign(%=)

**16.Explain JS If Statement.**

* The JavaScript if-else statement is used to execute the code whether the specified condition is true or false. There are three forms of if statement in JavaScript.
  1. JavaScript If statement

-It evaluates the content only if condition is true.

- if(condition){ //content to be evaluated }

* 1. Javascript If…else statement

-It evaluates the content whether condition is true of false.

-if(expression){ //content to be evaluated if condition is true }

else{ //content to be evaluated if condition is false }

* 1. Javascript If…else….if statement

-It executes code only if condition is true from several condition.

-if(expression1){ //code executed if expression1 is true }

else if(expression2){ // code executed if expression2 is true }

else if(expression3){ // code executed if expression3 is true }

else{ // code executed if no expression is true }

**17.Explain JS Switch statement.**

* The switch statement is used to perform different actions based on different conditions.
* The switch expression is evaluated once. The value of the expression is compared with the values of each case. If there is a match, the associated block of code is executed. If there is no match, the default code block is executed.
* Break will stop the execution inside the switch block. It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.
* The default keyword specifies the code to run if there is no case match. The default case does not have to be the last case in a switch block

switch(expression) {  
  case x:  
    *// code block*    break;  
  case y:  
    *// code block*    break;  
  default:  
    // code block  
}

**18.Explain JS Loops.**

* Loops can be used to execute a block of code a number of times.
* Loops are handy, if you want to run the same code over and over again, each time with a different value.

for (let i = 0; i < cars. Length; i++) {  
  text += cars[i] + "<br>";  
}

const person = {fname:"John", lname:"Doe", age:25};

// const numbers = [45, 4, 9, 16, 25];   
let text = "";  
for (let x in person) {  
  text += person[x];  
}

const numbers = [45, 4, 9, 16, 25];  
let txt = "";  
numbers.forEach(myFunction);  
  
function myFunction(value, index, array) {  
  txt += value;  
}

const cars = ["BMW", "Volvo", "Mini"];

//let language = "JavaScript"; :-Looping over a string  
let text = "";  
for (let x of cars) {  
  text += x;  
}

while (i < 10) {  
  text += "The number is " + i;  
  i++;  
}

* JavaScript supports different kinds of loops:

for - loops through a block of code a number of times

for/in - loops through the properties of an object

for/of - loops through the values of an iterable object

while - loops through a block of code while a specified condition is true

forEach()- method calls a function (a callback function) once for each array element.

do/while - also loops through a block of code while a specified condition

is true.

**19.Explain JS Function.**

* A JavaScript function is a block of code designed to perform a particular task.
* A JavaScript function is executed when "something" invokes it (calls it).
* A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses (arguments). Then follow the {codes to be executed}.
* Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
* Function **arguments** are the **values** received by the function when it is invoked.
* When JavaScript reaches a return statement, the function will stop executing. If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement. Functions often compute a **return value**. The return value is "returned" back to the "caller".
* Functions can be used as variable value :

let text = "The temperature is " + toCelsius(77) + " Celsius";

* A function is invoked when :

-When an event occurs (when a user clicks a button)

-When it is invoked (called) from JavaScript code

-Automatically (self invoked):

A self-invoking expression is invoked (started) automatically, without being called. Function expressions will execute automatically if the expression is followed by ().

(function () {  
  let x = "Hello!!";  // I will invoke myself  
})();

**20.Explain why we use Function.**

* You can reuse code: Define the code once, and use it many times.
* You can use the same code many times with different arguments, to produce different results.
* It makes our program compact. We don’t need to write many lines of code each time to perform a common task.

**21.Explain Function Constructor.**

* In JavaScript, the purpose of Function constructor is to create a new Function object.
* It executes the code globally.
* , a function is created dynamically but in an unsecured way.

new Function ([arg1[, arg2[, ....arg3]],] functionBody func def)

ex:

var add=new Function("num1","num2","return num1+num2");

document.getElementById(add(2,5));

**22.Explain Anonymous Function.**

* A Anonymous function is a function without name.
* Functions stored in variables do not need function names. They are always invoked (called) using the variable name.

A JavaScript function can also be defined using an expression. A function expression can be stored in a variable

const x = function (a, b) {return a \* b}; //a func expression  
let z = x(4, 3);

document.getElementById(“demo”).innerHTML=z

**23.Explain Function Methods.**

* The toString () method returns the function as a string.

function myFunction (a, b) {  
  return a \* b;  
}  
document.getElementById(“id”).innerHTML= myFunction.toString();

ANS : function myFunction(a, b) { return a \* b; }

* The call() method can be used to invoke (call) a method with an owner object as an argument (parameter). With it, an object can use a method belonging to another object. The call() method can accept arguments. It take arguments sepatrately. It is used to call a function contains this value and an argument list.

person.fullName.call(person1,"Oslo", "Norway");

* The apply() method is similar to the call() method but it take argument as an Array. It is used to call a function contains this value and a single array of arguments.

person.fullName.apply(person1, ["Oslo", "Norway"]);

* Since JavaScript **arrays** do not have a max() method, you can apply the Math.max() method with apply() to find the largest number (in a list of numbers).
* With the bind() method, an object can borrow a method from another object.

**24.What is this.**

* The this keyword refers to different objects depending on how it is used.
* In a function, this refers to the **global object**.
* In an event, this refers to the **element** that received the event.
* Methods like call(), apply(), and bind() can refer this to **any object**.
* this is not a variable. It is a keyword. You cannot change the value of this.

**25.What is Closure.**

* Closure in action that inner function can have access to outer function variable and parameter and as well as global variables.
* A closure is a function having access to the parent scope, even after the parent function has closed.

  <script>

        var a=10;

        function outerFunction(){

            var b=5;

            function innerFunction(){

                var c=20;

                document.getElementById("demo").innerHTML=a+b+c;

            }

            innerFunction();

        }

        outerFunction();

    </script>

**26.What is scope chain and Lexical Scoping.**

* The scope chain is used to resolve the value of variable names in Javascript. Without a scope chain, the Javascript engine wouldn't know which value to pick for a certain variable name if there are multiply defined at different scopes. The scope chain in Javascript is lexically defined, which means that we can see what the scope chain will be by looking at the code.

<script>

        var a=10;

        function outerFunction(){

            var b=5;

            function innerFunction(){

                var c=20;

                document.getElementById("demo").innerHTML=a+b+c;

            }

            innerFunction();

        }

        outerFunction();

    </script>

**27.What is JS Object.**

* A Javascript object is an entity having state and behaviour (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.
* A JavaScript object is a collection of **named values.** The named values, in JavaScript objects, are called **properties**.
* It is a common practice to declare objects with the const keyword.

const car = {type:"Fiat", model:"500", color:"white"};

* You can access object properties in two ways: *objectName.propertyName :car.type*

*objectName["propertyName"] :car[‘type’]*

**28.What is JS Object Method.**

* Objects can also have **methods**.
* Methods are **actions** that can be performed on objects.
* Methods are stored in properties as **function definitions**.
* You access an object method with the following syntax:

*objectName.methodName()*

const person = {  
  firstName: "John",  
  lastName : "Doe",  
  id       : 5566,  
  fullName : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};document.getElementById("demo").innerHTML = person.fullName();

* Adding a method to an object

person.name = function () {  
  return this.firstName + " " + this.lastName;  
};

**29.How we create JS Objects.**

* By object literal : An object literal is a list of name:value pairs (like age:50) inside curly braces {}.

var emp={id:102,name:"Shyam Kumar",salary:40000}

* By creating instance of Object directly (using new keyword)

var emp=new Object();

emp.id=101;

emp.name="Ravi Malik";

emp.salary=50000;

* By using an object constructor (using new keyword)

function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;

}

e=new emp(103,"Vimal Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);

* Create an object using Object.create().

**30.How to add/delete new properties in Object.**

* You can add new properties to an existing object by simply giving it a value.

const person = {

            firstname: "John",

            lastname: "Doe",

            age: 50,

            eyecolor: "blue"

        };

        person.nationality = "English";

delete person.age/ delete person["age"];

        document.getElementById("demo").innerHTML =

            person.firstname + " is " + person.nationality + ".";

* The delete keyword deletes a property from an object:

**31.How To Diaplay JS Objects.**

* Displaying the Object Properties by name.
* Displaying the Object Properties in a Loop
* Displaying the Object using Object.values(). Any JavaScript object can be converted to an array using Object.values(objectname).
* Displaying the Object using JSON.stringify(). Any JavaScript object can be stringified (converted to a string) with the JavaScript function JSON.stringify(objectname)

**32.Explain JS Accessors(Getters and Setters).**

* Accessor properties are represented by “getter” and “setter” methods that are allow you to define Object Accessors (Computed Properties). In an object literal they are denoted by get and set
* Sometimes it is desirable to allow access to a property that returns a dynamically computed value, or you may want to reflect the status of an internal variable without requiring the use of explicit method calls. In JavaScript, this can be accomplished with the use of a getter.
* It is not possible to simultaneously have a getter bound to a property and have that property actually hold a value, although it is possible to use a getter and a setter in conjunction to create a type of pseudo-property.
* In JavaScript, a setter can be used to execute a function whenever a specified property is attempted to be changed. Setters are most often used in conjunction with getters to create a type of pseudo-property. It is not possible to simultaneously have a setter on a property that holds an actual value.
* Why Using Getters and Setters?

It gives simpler syntax. It allows equal syntax for properties and methods. It can secure better data quality. It is useful for doing things behind-the-scenes

**33.Explain JS object Constructors.**

* Sometimes we need a "blueprint" for creating many objects of the same "type". The way to create an "object type", is to use an object constructor function.
* Objects of the same type are created by calling the constructor function with the new keyword.
* In a constructor function this does not have a value. It is a substitute for the new object. The value of this will become the new object when a new object is created.
* Note: It is considered a good practice to capitalize the first letter of your constructor function.
* The constructor property returns a reference to the Object constructor function that created the instance object. Note that the value of this property is a reference to the function itself, not a string containing the function's name.

**34.Explain JS Prototypes.**

* All JavaScript objects inherit properties and methods from a prototype.

Date objects inherit from Date.prototype

Array objects inherit from Array.prototype

Person objects inherit from Person.prototype

* The Object.prototype is on the top of the prototype inheritance chain:
* Date objects, Array objects, and Person objects inherit from Object.prototype.
* The JavaScript prototype property allows you to add new properties/methods to object constructors:

**35.Explain JS Itearators.**

* It simply means looping over a sequence of elements.
* Iterable objects are a generalization of arrays. That’s a concept that allows us to make any object useable in a for..of loop.
* Iterable objects are objects that can be iterated over with for..of.
* Of course, Arrays are iterable. But there are many other built-in objects, that are iterable as well. For instance, strings are also iterable.
* If an object isn’t technically an array, but represents a collection (list, set) of something, then for..of is a great syntax to loop over it

**37.Explain JS Array and how to create it.**

* An array is a special variable, which can hold more than one value under a single name, and you can access the values by referring to an index number.
* Using an array literal is the easiest way to create a JavaScript Array.

const array\_name = [item1, item2, ...];

* You can also create an array, and then provide the elements:

const cars = [];  
cars[0]= "Saab";  
cars[1]= "Volvo";  
cars[2]= "BMW";

* Using the JavaScript Keyword new, We also create a Array.

const cars = new Array("Saab", "Volvo", "BMW");

const points = new Array(40, 100, 1, 5, 25, 10) is same as  
const points = [40, 100, 1, 5, 25, 10];

* But while creating a array with one element follow [] method instead new keyword

const points = new Array(40) not same aa const points = [40];

* For simplicity, readability and execution speed, use the array literal method.
* You access an array element by referring to the index number:

const cars = ["Saab", "Volvo", "BMW"];  
document.getElementById(“demo”).innerHTML=cars[0];

* With JavaScript, the full array can be accessed by referring to the array name

const cars = ["Saab", "Volvo", "BMW"];  
document.getElementById("demo").innerHTML = cars;

* Changing an Array Element:

const cars = ["Saab", "Volvo", "BMW"];  
cars[0] = "Opel";

document.getElementById("demo").innerHTML = cars;

* Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.
* In the JavaScript operator typeof returns "object" for a Array. To identify that This is a Array we can use a new method Array.isArray() which is defined in ES6 or by using the instanceof operator returns true if an object is created by a given constructor:

const cars = ["Saab", "Volvo", "BMW"];  
document.getElementById(“demo”).innerHTML=Array.isArray(cars);

// document.getElementById(“demo”).innerHTML=cars.instanceof.Array

**37.Explain JS Array Properties.**

* The length property of an array returns the length of an array (the number of array elements).
* The easiest way to add a new element to an array is using the push() method. New element can also be added to an array using the length property:

const fruits = ["Banana", "Orange", "Apple"];  
fruits.push("Lemon") // fruits[fruits.length] = "Lemon";

document.getElementById("demo").innerHTML = fruits;

**38.Explain JS associative Array.**

* Many programming languages support arrays with named indexes. Arrays with named indexes are called associative arrays (or hashes).
* JavaScript does not support arrays with named indexes. In JavaScript, arrays always use numbered indexes.

**39.Explain JS Array Methods.**

**40.Explain JS Math.**

* The JavaScript Math object allows us to perform mathematical tasks on numbers.
* Unlike other objects, the Math object has no constructor.
* The Math object is static.
* All methods and properties can be used without creating a Math object first.
* The syntax for any Math property is : Math.property.
* JavaScript provides 8 mathematical constants that can be accessed as Math properties:
* Math.E        // returns Euler's number  
  Math.PI       // returns PI  
  Math.SQRT2    // returns the square root of 2  
  Math.SQRT1\_2  // returns the square root of 1/2  
  Math.LN2      // returns the natural logarithm of 2  
  Math.LN10     // returns the natural logarithm of 10  
  Math.LOG2E    // returns base 2 logarithm of E  
  Math.LOG10E   // returns base 10 logarithm of E

**41.Explain JS Math Methods.**

* The syntax for Math any methods is : Math.methodname(number).
* Math.round(x) returns the nearest integer: Math.round(4.6)//5
* Math.ceil(x) returns the value of x rounded **up** to its nearest integer: Math.ceil(4.9);//5
* Math.floor(x) returns the value of x rounded **down** to its nearest integer: Math.floor(4.9);//4
* Math.trunc(x) returns the integer part of x: Math.floor(4.9);//4
* Math.sign(x) returns 1 if x is negative or positive & 0 if x is null
* Math.pow(x, y) returns the value of x to the power of y: Math.pow(8, 2);
* Math.sqrt(x) returns the square root of x: Math.sqrt(16);//4
* Math.abs(x) returns the absolute (positive) value of x:

Math.abs(-4.7);//4.7

* Math.sin(x)// Math.cos(x) returns the sine// cosine (a value between -1 and 1) of the angle x (given in radians).
* If you want to use degrees instead of radians, you have to convert degrees to radians. Angle in radians = Angle in degrees x PI / 180.

Math.sin(90 \* Math.PI / 180);//1 as sin90=1

* Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments

Math.min//max(0, 150, 30, 20, -8, -200);

* Math.log(x) returns the natural logarithm of x. The natural logarithm returns the time needed to reach a certain level of growth:
* Math.random() returns a random number between 0 (inclusive),  and 1 (exclusive). Math.random() always returns a number lower than 1.
* Math.random() used with Math.floor() can be used to return random integers.

Math.floor(Math.random() \* 10);

**42.Explain JS Booleans.**

* Very often, in programming, you will need a data type that can only have one of two values, like yes/no, on/off, true/false. For this, JavaScript has a **Boolean** data type. It can only take the values **true** or **false**.
* You can use the Boolean() function to find out if an expression (or a variable) is true: Boolean(10 > 9)/ (10 > 9)/ 10 > 9
* Everything With a "Value" is True except 0
* Everything Without a "Value" is False. The Boolean value of undefined is false.
* The Boolean value of **""**(empty string) is **false**
* The Boolean value of **-0** (minus zero) is **false**:
* The Boolean value of **null/false/NaN** is **false**

**43.Explain JS Break/Continue statement.**

* The break statement was used to "jump out" of a switch() statement.
* The break statement can also be used to jump out of a loop:

for (let i = 0; i < 10; i++) {  
  if (i === 3) { break; }  
  text += "The number is " + i + "<br>";  
} // It will stop executing when it reaches i===3

* The continue statement breaks/skips one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

for (let i = 0; i < 10; i++) {  
 if (i === 3) { continue; }//it will skip the execution when i==3  
  text += "The number is " + i + "<br>";  
}

* The break and the continue statements are the only JavaScript statements that can "jump out of" a code block.

**44.Explain JS Labels.**

* To label JavaScript statements you precede the statements with a label name and a colon.
* The break and the continue statements are the only JavaScript statements that can "jump out of" a code block.
* The continue statement (with or without a label reference) can only be used to skip one loop iteration. The break statement, without a label reference, can only be used to jump out of a loop or a switch. With a label reference, the break statement can be used to jump out of any code block:

const cars = ["BMW", "Volvo", "Saab", "Ford"];  
list: {  
  text += cars[0] + "<br>";  
  text += cars[1] + "<br>";  
  break list;  
  text += cars[2] + "<br>";  
  text += cars[3] + "<br>";

**45.Explain JS Set.**

* A JavaScript Set is a collection of unique values. Each value can only occur once in a Set.
* You can create a JavaScript Set by:

Passing an Array to new Set()

Create a new Set and use add() to add values

Create a new Set and use add() to add variables

* Set property size returns the number of elements in a Set

Doc……=setname.size

**46.Explain JS Set Methods.**

* Pass an Array to the new Set() constructor. Or Create a Set and add values:

const letters = new Set(["a","b","c"]);

const letters = new Set();  
// Add Values to the Set  
letters.add("a");  
letters.add("b");  
letters.add("c");

* // Create a Set and variable  
  const letters = new Set();  
    
  // Create Variables  
  const a = "a";  
  const b = "b";  
  const c = "c";  
    
  // Add Variables to the Set  
  letters.add(a);  
  letters.add(b);  
  letters.add(c);
* The add() Method adds a value. If you add equal elements, only the first will be saved: letters.add("d");letters.add("e");
* The forEach() method invokes (calls) a function for each Set element:

// Create a Set  
const letters = new Set(["a","b","c"]);  
  
// List all Elements  
let text = "";  
letters.forEach (function(value) {  
  text += value;  
})

* The values() method returns a new iterator object containing all the values in a Set: letters.values()
* Now you can use the Iterator object to access the elements:

// List all Elements  
let text = "";  
for (const x of letters.values()) {  
  text += x;

**47.Explain JS Map Methods and how to create it.**

* A Map holds key-value pairs where the keys can be any datatype.
* A Map remembers the original insertion order of the keys.
* You can create a JavaScript Map by:Passing an Array to new Map() &&

Create a Map and use  set() mrthod

* You can create a Map by passing an Array to the new Map() constructor:

const fruits = new Map([  
  ["apples", 500],  
  ["bananas", 300],  
  ["oranges", 200]  
]);

* You can add elements to a Map with the set() method:

// Create a Map  
const fruits = new Map();  
  
// Set Map Values  
fruits.set("apples", 500);  
fruits.set("bananas", 300);  
fruits.set("oranges", 200);

The set() method can also be used to change existing Map values:

fruits.set("apples", 200);

* The get() method gets the value of a key in a Map:

fruits.get("apples");    // Returns 500

* The size property returns the number of elements in a Map:
* The delete() method removes a Map element: fruits.delete("apples");
* The has() method returns true if a key exists in a Map:

fruits.has("apples");

The forEach() method calls a function for each key/value pair in a Map:

// List all entries  
let text = "";  
fruits.forEach (function(value, key) {  
  text += key + ' = ' + value;  
})

* The entries() method returns an iterator object with the [key, values] in a Map:

// List all entries  
let text = "";  
for (const x of fruits.entries()) {  
  text += x;

**46.Explain JS Errors.**

* When executing JavaScript code, different errors can occur.
* Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.
* When these errors are occurred, we can generate custom error messages by using try, catch, finally and throw statements and we can control program flow
* The try statement defines a code block to run (to try).
* The catch statement defines a code block to handle any error.
* The finally statement defines a code block to run regardless of the result.
* The throw statement defines a custom error.
* **Remains……**

**47.Explain JS Scope.**

* Scope determines the accessibility (visibility) of variables.
* JavaScript has 3 types of scope: Block scope, Function scope, Global scope

**48.Explain JS Hosting.**

* Hoisting is JavaScript's default behavior of moving declarations to the top of the current scope (to the top of the current script or the current function).
* In JavaScript, a variable can be declared after it has been used. In other words; a variable can be used before it has been declared.
* If a developer doesn't understand hoisting, programs may contain bugs (errors).To avoid bugs, always declare all variables at the beginning of every scope.
* Variables defined with let and const are hoisted to the top of the block, but not initialized. Meaning: The block of code is aware of the variable, but it cannot be used until it has been declared.
* Using a let/const variable before it is declared will result in a ReferenceError. The variable is in a "temporal dead zone" from the start of the block until it is declared:

carName = "Volvo";  
let/const carName;

* JavaScript only hoists declarations, not initializations.

var x = 5; // Initialize x  
  
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x + " " + y;           // Display x and y  
  
var y = 7; // Initialize y //show wrong result

**49.Explain JS Modules.**

* JavaScript modules allow us to break up our code into separate files.
* This makes it easier to maintain the code-base.
* JavaScript modules rely on the import and export statements.
* You can export a function or variable from any file. There are two types of exports: Named and Default. You can create named exports two ways. Export In-line individually, or Export all at once at the bottom.

Individually all at once

export const name = "Jesse"; const name = "Jesse";  
export const age = 40; const age = 40;

export {name, age};

* Let us create another file, named message.js, and use it for demonstrating default export. You can only have one default export in a file.
* W e can import modules into a file in two ways, based on if they are named exports or default exports. Named exports are constructed using curly braces. Default exports are not.
* Modules only work with the HTTP(s) protocol.

**50.Explain JS “use strict”.**

* "use strict"; Defines that JavaScript code should be executed in "strict mode".
* It is not a statement, but a literal expression, ignored by earlier versions of JavaScript before ECMAScript version 5.
* With strict mode, you can not, for example, use undeclared variables/objects, cannot delete a var/obj and cannot duplicate a parameter name.
* You can use strict mode in all your programs. It helps you to write cleaner code, like preventing you from using undeclared variables
* Strict mode is declared by adding "use strict"; to the beginning of a script or a function.
* Declared at the beginning of a script, it has global scope (all code in the script will execute in strict mode): "use strict"; x = 3.14;

**51.Why we use JS use strict.**

* Strict mode makes it easier to write "secure" JavaScript.
* Strict mode changes previously accepted "bad syntax" into real errors.

As an example, in normal JavaScript, mistyping a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable.

* In normal JavaScript, a developer will not receive any error feedback assigning values to non-writable properties. In strict mode, any assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object, will throw an error.

**52.What are the future reserved JS Keywords.**

* Implements, interface, let, package, private, protected, public, static, yield

**53.What is JS Class.**

* ECMAScript 2015, also known as ES6, introduced JavaScript Classes.
* A JavaScript class is not an object. JavaScript Classes are templates for JavaScript Objects.
* Syntax: Use the keyword class to create a class. Always add a predefined method named constructor().Then add any number of user defined methods.

class ClassName {  
  constructor() { ... } method1() { ... } method2() { ... }  
}

* When you have a class, you can use the class to create objects.

The example below uses the **Car class** to create two **Car objects**.

<script>

class Car {

  constructor(name, year) {

    this.name = name;

    this.year = year;

    }

}

const myCar = new Car("Ford",2014);

document.getElementById("demo").innerHTML =

myCar.name + " " + myCar.year;

</script>

* The constructor method is called automatically when a new object is created.
* Classes also allows you to use getters and setters. It can be smart to use getters and setters for your properties, especially if you want to do something special with the value before returning them, or before you set them. To add getters and setters in the class, use the get and set keywords.

**54.What is JS Class constructer method.**

* The constructor method is a special method: It has to have the exact name "constructor".
* It is executed automatically when a new object is created
* It is used to initialize object properties
* If you do not define a constructor method, JavaScript will add an empty constructor method.

**54.What is JS Class Inheritance.**

* A class created with a class inheritance inherits all the methods from another class
* To create a class inheritance, use the extends keyword.
* Now we call The super() method refers to the parent class.
* By calling the super() method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.
* Inheritance is useful for code reusability: reuse properties and methods of an existing class when you create a new class.

**55.What is Callback function.**

* A callback is a function passed as an argument to another function
* This technique allows a function to call another function
* A callback function can run after another function has finished.
* In the real world, callbacks are most often used with asynchronous functions, where one function has to wait for another function (like waiting for a file to load).

**56.What is Asynchronous function.**

* Functions running in parallel with other functions are called asynchronous
* A good example is JavaScript setTimeout(). When using the JavaScript function setTimeout(), you can specify a callback function to be executed on time-out.
* When using the JavaScript function setInterval(), you can specify a callback function to be executed for each interval:

**57.What is Promise Function.**

* A “producing code” that does something and takes some time. For instance, some code that loads the data over a network.
* A “consuming code” that wants the result of the “producing code” once it’s ready and it must wait for the result. Many functions may need that result.
* A promise is a special JavaScript object that links the “producing code” and the “consuming code” together.
* A JavaScript Promise object can be: Pending, Fulfilled, Rejected
* The Promise object supports two properties: **state** and **result**.
* While a Promise object is "pending" (working), the result is undefined.
* When a Promise object is "fulfilled", the result is a value.
* When a Promise object is "rejected", the result is an error object.
* You cannot access the Promise properties state and result. You must use a Promise method to handle promises.
* Promise.then() takes two arguments, a callback for success and another for failure. Both are optional, so you can add a callback for success or failure only.

**58.What is Async/Await.**

* "async and await make promises easier to write". async makes a function return a Promise. await makes a function wait for a Promise
* The keyword async before a function makes the function return a promise:
* The keyword await before a function makes the function wait for a promise. The await keyword can only be used inside an async function.

**1.What is Javascript.**

* HTML has no processing capacity. It has no feature to take input from user and do the processing and then will give the output. It only capacity of took the input and gave output. So HTML needs helps from Javascript which is a processing language for processing and after processing JS give the output to HTML and HTML send that output to the browser and the user.
* JS cannot deal with the browser directly like cannot take the input from user directly. It took through HTML.
* Its also called as Processing/Scripting/Client-side Processing Language.

**2.What is Databinding.**

* The process of sending and receiving data from HTML to Javascript or Javascript to HTML is called as Databinding.

**3.Explain document.getElementById(“idname”).innerHTML.**

* **.document** Represent your current Page/file.
* **.getElementById** will help you to search an element whose id is **“idname”**. It is a predefined function of JS DOM library.
* **.innerHTML** is represent the content of an html element.

**4.What happened after clicking button which have a JS Function.**

* JS send some data to html

**5.if function**

1. Condition help us to control the dataflow in project

if(a>b){

        console.log(a,b,"The higher value is : ",a)

    }else{console.log(a,b,"The higher value is : ",b)}

If the condition(a>b) is true then print the statement a,b"The higher value is : ",a.

If the condition is false then print the statement(a,b"The higher value is: ",b)

2.If condition is help you to check range value(a>b) also and exact value also(a==b).But using Switch case you can check only exact value.

For Loop:

if you want to run the same code over and over again, each time with a different value. Loops can execute a block of code a number of times.

for(var i=1;i<10;i++){

        console.log('The odd number : ',i); //Code Block

    }

1. Here we initialize a variable { var i=1;} or  sets a variable before the loop starts (var i = 0).
2. Then specify the condition for the loop to run (i must be less than 10).
3. We use increment operator (i++) to Increase the value of i with 1, each time after the code block in the loop has been executed.
4. Then we check the condition till its false and print the data in the browser.

While Loop:

The while loop loops through a block of code as long as a specified condition is true.

var i=1;

    while(i<10){

       console.log("While Say : " + i);

        i++;

    }

1. Here we initialize a variable { var i=1;} or  sets a variable before the loop starts (var i = 0).
2. Then specify the condition for the loop to run (i must be less than 10).
3. Then we run the statement Or execute the code blocks till the condition is true.
4. We use increment operator (i++) to Increase the value of i with 1, each time after the code block in the loop has been executed. Or after we run the statement
5. If in do while loop the condition does not mentioned,Then the loop function runs infinite times.

Map Function:

On Array once you run the map(),It will read individual element and after read it will split the element in two parts(value,index).After that use fat arrow method inside it to print the data.

Here product is not a array but it holds the data on form of array.

var item=[

        {catagory:"Books", product:["Java","PHP","Node JS","React JS"]}]

Events:

Event will happen when mouse enter on a button or click a button. Then Javascript run the function defined inside the button and execute a series of codes and send data to the div having id ‘demo’

document.getElementById("demo").innerHTML = "Hello JavaScript!"

The example above "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript"