### **1) What is JavaScript?**

**JavaScript** is a scripting language. It is different from Java language. It is object-based, lightweight, cross-platform translated language. It is widely used for client-side validation.

### **List some features of JavaScript.**

* Lightweight
* Complementary to Java
* Complementary to HTML
* Open source Cross-platform

### **List some of the advantages of JavaScript.**

* Server interaction is less
* Feedback to the visitors is immediate
* Interactivity is high
* Interfaces are richer

### **List some of the disadvantages of JavaScript.**

Some of the disadvantages of JavaScript are:

* No support for multithreading
* No support for multiprocessing
* Reading and writing of files is not allowed
* No support for networking applications.

### **Define a named function in JavaScript.**

The function which has named at the time of definition is called a named function. For example

1. function msg()
2. {
3. document.writeln("Named Function");
4. }
5. msg();

### **In JavaScript what is an argument object?**

The variables of JavaScript represent the arguments that are passed to a function.

### **Define closure.**

In JavaScript, we need closures when a variable which is defined outside the scope in reference is accessed from some inner scope.

1. var num = 10;
2. function sum()
3. {
4. document.writeln(num+num);
5. }
6. sum();

### **If we want to return the character from a specific index which method is used?**

The JavaScript string charAt() method is used to find out a char value present at the specified index.

1. var str="Javatpoint";
2. document.writeln(str.charAt(4));

### **What is BOM?**

**BOM** stands for Browser Object Model. It provides interaction with the browser. The default object of a browser is a window. the window object provides various properties like document, history, screen, navigator, location, innerHeight, innerWidth,

### **What is the difference between == and ===?**

The == operator checks equality only whereas === checks equality, and data type, i.e., a value must be of the same type

### **28) How to create objects in JavaScript?**

There are 3 ways to create an object in JavaScript.

1. By object literal
2. By creating an instance of Object
3. By Object Constructor

Let's see a simple code to create an object using object literal.

1. emp={id:102,name:"Rahul Kumar",salary:50000}

### **) How to create an array in JavaScript?**

There are 3 ways to create an array in JavaScript.

1. By array literal
2. By creating an instance of Array
3. By using an Array constructor

Let's see a simple code to create an array using object literal.

1. var emp=["Shyam","Vimal","Ratan"];

### **What does the isNaN() function?**

The isNan() function returns true if the variable value is not a number. For example:

1. function number(num) {
2. if (isNaN(num)) {
3. return "Not a Number";
4. }
5. return "Number";
6. }
7. console.log(number('1000F'));
8. // expected output: "Not a Number"
10. console.log(number('1000'));
11. // expected output: "Number"

### **In which location cookies are stored on the hard disk?**

The storage of cookies on the hard disk depends on the OS and the browser.

### **What's the difference between event.preventDefault() and event.stopPropagation() methods in JavaScript?**

In JavaScript, the event.preventDefault() method is used to prevent the default behavior of an element.

**For example:** If you use it in a form element, it prevents it from submitting. If used in an anchor element, it prevents it from navigating. If used in a contextmenu, it prevents it from showing or displaying.

On the other hand, the event.stopPropagation() method is used to stop the propagation of an event or stop the event from occurring in the bubbling or capturing phase.

### **What is the difference between undefined value and null value?**

**Undefined value:** A value that is not defined and has no keyword is known as undefined value. For example:

1. int number;//Here, a number has an undefined value.

**Null value:** A value that is explicitly specified by the keyword "null" is known as a null value. For example:

1. String str=null;//Here, str has a null value.

### **What is the difference between View state and Session state?**

"View state" is specific to a page in a session whereas "Session state" is specific to a user or browser that can be accessed across all pages in the web application.

### **How can we detect OS of the client machine using JavaScript?**

The **navigator.appVersion** string can be used to detect the operating system on the client machi

### **What is this keyword in JavaScript?**

The this keyword is a reference variable that refers to the current object. For example:

1. var address=
2. {
3. company:"Javatpoint",
4. city:"Noida",
5. state:"UP",
6. fullAddress:function()
7. {
8. return this.company+" "+this.city+" "+this.state;
9. }
10. };
11. var fetch=address.fullAddress();
12. document.writeln(fetch);

### **What is the requirement of debugging in JavaScript?**

* Using console.log() method
* Using debugger keyword

### 

### **What is the use of a TypedArray object in JavaScript?**

JavaScript typed arrays are array-like objects that **provide a mechanism for reading and writing raw binary data in memory buffers**. Array objects grow and shrink dynamically and can have any JavaScript value.

1. function display()
2. {
3. var arr1= [1,2,3,4,5,6,7,8,9,10];
4. arr1.copyWithin(2) ;
5. document.write(arr1);
6. }
7. display();

### **What is the use of a Set object in JavaScript?**

Set object is used to store the elements with unique values. The values can be of any type i.e. whether primitive values or object references. For example:

1. function display()
2. {
3. var set = new Set();
4. set.add("jQuery");
5. set.add("AngularJS");
6. set.add("Bootstrap");
7. for (let elements of set) {
8. document.writeln(elements+"**<br>**");
9. }
10. }
11. display();

### **What is the use of a WeakSet object in JavaScript?**

WeakSet object is the type of collection that allows us to store weakly held objects. Unlike Set, the WeakSet are the collections of objects only. It doesn't contain the arbitrary values. For example:

function display(){

var ws = new WeakSet();

var obj1 = {};

var obj2 = {};

ws.add(obj1);

ws.add(obj2);

document.write(ws.has(obj1))

document.write(ws.has(obj2))

}

display()

**Reverse string**

function reverseString(*str*){

  return str.split("").reverse().join("");

}

console.log(reverseString("hyderbad"))

**Merge array and display in for loop**

let a = [1,2,3,4];

let b = [5,6,7,8];

let c = a.concat(b);

let item = '<ul>'

for(i=0; i<c.length; i++){

  item+= '<li>' + c[i] + '</li>'

}

item += '</ul>'

document.getElementById('demo').innerHTML = item

**Get input value**

 var text = document.getElementById("name");

            text.addEventListener('keyup', function(){

                document.getElementById('demo').innerHTML = text.value

            })

**Self Invoking Funcion**

A self-invoking expression is **invoked (started) automatically, without being called**. Function expressions will execute automatically if the expression is followed by (). You cannot self-invoke a function declaration.

(function(){

  console.log("hello world");

})()

Out put Hello world

**Hoisting**

JavaScript where all the variable and function declarations are moved on top.. The scope can be both local and global.

x =15;

console.log(x);

var x;

**Strict Mode**

Using strict mode javascript silent errors can be easily detected as they would throw an error. This makes javascript debugging much easy and helps developers to avoid unnecessary mistakes

“use strict”

**Let Keyword;**

Variables defined with let cannot be Redeclared.

Variables defined with let must be Declared before use

Variables defined with let have Block Scope.

let x = "John Doe";

let x = 0;

*// SyntaxError: 'x' has already been declared*

**Cosnt:**

Variables defined with const cannot be Redeclared.

Variables defined with const cannot be Reassigned.

Variables defined with const have Block Scope

const name = "suresh";

name ="mahesh"

**javascritpt are dynamic type;**

JavaScript has dynamic types. This means that the same variable can be used to hold different data types

let x =20;

 x="john";

 console.log(x)

**function**

Funcions

function User(*a*,*b*){

  return a\*b

}

console.log(User(4,5))

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).

Function Return

When JavaScript reaches a return statement, the function will stop executing.

**Why 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.

**Local Variable**

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables can only be accessed from within the function.

function myFunction() {

  let carName = "Volvo";

*// code here CAN use carName*

}

Javascript string

JavaScript strings are for storing and manipulating text.

let name = "vicky rocks";

console.log(name)

**currying in javascript;**

Currying is a function that takes one argument at a time and returns a new function expecting the next argument.

function add (*a*) {

  return function(*b*){

    return a + b;

  }

}

console.log(add(3)(4))

### **Passed by value and passed by reference**

* Passed By Values Are Primitive Data Types.

Consider the following example:

Here, the a=432 is a primitive data type i.e. a number type that has an assigned value by the operator.  When the var b=a code gets executed, the value of ‘var a’ returns a new address for ‘var b’ by allocating a new space in the memory, so that ‘var b’ will be operated at a new location.

Example:

var a = 432;

var b = a;

* Passed by References Are Non-primitive Data Types.

Consider the following example:

The reference of the 1st variable object i.e. ‘var obj’ is passed through the location of another variable i.e. ‘var obj2’ with the help of an assigned operator.

Example:

var obj = { name: "Raj", surname: "Sharma" };

var obj2 = obj;

### **Higher Order Functions**

Higher-order functions are the functions that take functions as arguments and return them by operating on other functions

function higherOrder(*fn*)

 {

  fn();

}

higherOrder(function() { console.log("Hello world") });

### **What is a WeakMap?**

Weakmap is referred to as an object having keys and values, if the object is without reference, it is collected as garbage.

### **What is Object Destructuring? (with examples)**

Object destructuring is a method to extract elements from an array or an object

const arr = [1, 2, 3];

const first = arr[0];

const second = arr[1];

const third = arr[2];

*//Example 2: Object Destructuring*

*//const arr = [1, 2, 3];*

*//const [first,second,third,fourth] = arr;*

console.log(first); *// Outputs 1*

console.log(second); *// Outputs 2*

console.log(third); *// Outputs 3*

### **What is a Temporal Dead Zone?**

Temporal Dead Zone is a behavior that occurs with variables declared using let and const keywords before they are initialized.

### **How do you empty an array in JavaScript?**

There are a few ways in which we can empty an array in JavaScript:

*//By assigning array length to 0:*

var arr = [1, 2, 3, 4];

arr.length = 0;

*//By assigning an empty array:*

var arr = [1, 2, 3, 4];

arr = [];

*//By popping the elements of the array:*

var arr = [1, 2, 3, 4];

while (arr.length > 0) {

arr.pop();

}

*//By using the splice array function:*

var arr = [1, 2, 3, 4];

arr.splice(0, arr.length);

console.log(arr)

#### **What are Event Bubbling and Event Capturing?**

* **Event Capturing** – Also known as trickling, Event Capturing is rarely used. The process starts with the outermost element capturing the event and then propagating it to the innermost element.
* **Event Bubbling** – In this process, the event gets handled by the innermost element first and then propagated to the outermost element.

Call():With the apply() method, you can write a method that can be used on different objects.

Apply():

Bind():

Call and apply methods same it will take argument separately and array

const person = {

  fullName: function(*city*, *state*){

    return this.fname + " " + this.lname + " "+ city +state;

  }

}

const person1 = {

  fname: "suresh",

  lname:"gurram"

}

let result = person.fullName.apply(person1, ["bangalore" ,"ka"])

console.log(result)

Bind()With the bind() method, an object can borrow a method from another object.

const person = {

  fname:'suresh',

  lname: "gurram",

  fullName : function(){

    return this.fname + " " + this.lname

  }

}

const person1 = {

  fname : "john",

  lname : 'doe',

}

let fullName = person.fullName.bind(person1);

console.log(fullName())

**Callback Funciton**

Def: function passed to another function as an argument is referred to as a callback function.

function Display(*data*){

console.log(data)

}

function add(*num1*, *num2*, *callback*){

  let sum = num1+num2;

  callback(sum)

}

let result = add(5,5, Display)

**Object Set()**

A JavaScript Set is a collection of unique values.Each value can only occur once in a Set.A Set can hold any value of any data type.

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

let text = "";

for(let x of letters.entries()){

    text += x

}

console.log(text)

**object map()**

A Map holds key-value pairs where the keys can be any datatype.A Map remembers the original insertion order of the keys.A Map has a property that represents the size of the map.

**Array map**

he map() method creates a new array by performing a function on each array element.

const arr = [1,2,3,4,9,8,6,5];

const arr1 = arr.map(myFunciton);

function myFunciton(value){

    return value \* 2

}

console.log(arr1)

**A sticky element toggles between relative and fixed , depending on the scroll position**. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

Static - this is the default value, all elements are in order as they appear in the document. Relative - the element is positioned relative to its normal position. Absolute - the element is positioned absolutely to its first positioned parent. Fixed - the element is positioned related to the browser window.

### **What is RWD?**

RWD stands for Responsive Web Design. This technique is used to display the designed page perfectly on every screen size and device, for example, mobile, tablet, desktop and laptop. You don't need to create a different page for each device.

The CSS box model is used to define the design and layout of elements of CSS.

* Margin - It removes the area around the border. It is transparent.
* Border - It represents the area around the padding
* Padding - It removes the area around the content. It is transparent.
* Content - It represents the content like text, images, etc.

Grid and flexbox. The basic difference between CSS Grid Layout and CSS Flexbox Layout is that **flexbox was designed for layout in one dimension - either a row or a column.** Grid was designed for two-dimensional layout - rows, and columns at the same time.

**The flex container properties are:**

* flex-direction.
* flex-wrap.
* flex-flow.
* justify-content.
* align-items.
* align-content.

**What is javascript prototype?**

**Prototypes are the mechanism by which JavaScript objects inherit features from one another**. In this article, we explain what a prototype is, how prototype chains work, and how a prototype for an object can be set.

const myObject = {

  city: "Madrid",

  greet() {

    console.log(`Greetings from ${this.city}`);

  },

};

myObject.greet(); *// Greetings from Madrid*

This is an object with one data property, city, and one method, greet(). If you type the object's name followed by a period into the console, like myObject., then the console will pop up a list of all the properties available to this object. You'll see that as well as city and greet, there are lots of other properties!

**Rest operator:** In functions when we require to pass arguments but were not sure how many we have to pass, the rest parameter makes it easier.

function sum(...*args*) {

  let sum = 0;

  for (let arg of args)

  sum += arg;

  return sum;

}

let x = sum(4, 9, 16, 25, 29, 100, 66, 77);

console.log(x)

**Spread operator:**The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.

var array1 = [10, 20, 30, 40, 50];

  var array2 = [60, 70, 80, 90, 100];

    var array3 = [...array1, ...array2];

   console.log(array3);

for object

const obj = {

        firstname: "Divit",

        lastname: "Patidar",

    };

    const obj2 = { ...obj };

    console.log(obj2);

## JavaScript Default Parameters

When no value is passed in the function definition, a default value is being set for the function definition, known as the **default parameter**. It allows the named parameters to get initialized with a default value when either undefined or no value is passed.

function sum(*a*=5, *b*=7)

{

    return a+b;

}

console.log(sum(5,7))

# **Destructuring assignment**

The two most used data structures in JavaScript are Object and Array.

* Objects allow us to create a single entity that stores data items by key.
* Arrays allow us to gather data items into an ordered list.

Destructuring assignment is a special syntax that allows us to “unpack” arrays or objects into a bunch of variables, as sometimes that’s more convenient.

Object Destructing

const num = {x: 100, y: 200};

const {x, y} = num;

console.log(x); *// 100*

console.log(y); *// 200*let arr = ["John", "Smith"]

Array destructing

let [firstName, surname] = arr;

console.log(firstName); *// John*

console.log(surname);  *// Smith*

**Arrow Function**

* This arrow function reduces lots of code and makes the mode more readable.
* Arrow function syntax automatically binds “this” to the surrounding code’s context.
* Writing the arrow **=>**is more flexible as compared with the writing **function**keyword.
* We may use arrow function syntax with our method associated with the array, like **map(), reduce(), filter().**

**Limitations**

* An arrow function doesn't have its own bindings with this or super.
* An Arrow function should not be used as methods.
* An arrow function can not be used as constructors.

### **Discuss the template literals in ES6.**

Template literals are a brand-new feature in ES6. It makes producing multiline strings and performing string interpolation simple.

### **What is Export Default and Named Export in ES6?**

* **Named Export:** Named exports are useful when one has to export multiple values. The name of the imported module must match that of the exported module.
* *//file rectangle.js*
* function perimeter(*x*, *y*) {
* return 2 \* (x + y);
* }
* function area(*x*, *y*) {
* return x \* y;
* }
* export { perimeter, area };
* *//while importing the functions in test.js*
* import { perimeter, area } from './rectangle;
* console.log(perimeter(4, 6)) *//20*
* console.log(area(4, 6)) *//24*
* **Default Export:**There is only one default export per module when it comes to default exports. A function, a class, an object, or anything else can be used as a default export. In default export, the naming of imports is fully autonomous, and we can choose any name we like.
* *// file module.js*
* var a = 6;
* export default a;
* *// test.js*
* *// while importing a in test.js*
* import b from './module';
* console.log(b);
* *// output will be 6*

### **What is the difference between for..of and for..in?**

* **for in**: runs over an object's enumerable property names.
* **for of**: (new in ES6) takes an object-specific iterator and loops through the data it generates.

Both the **for..of** and **for..in** commands iterate over lists, but the results they return are different: **for..in** returns a list of keys on the object being iterated, whereas **for..of** returns a list of values of the object's numeric attributes.

let arr = [3, 4, 5];

for (let i in arr) {

   console.log(i); *// "0", "1", "2",*

}

for (let i of arr) {

   console.log(i); *// "3", "4", "5"*

}

### **Explain Internationalization and Localization.**

These are JavaScript standard APIs that assist with operations such as collation, number formatting, currency formatting, and date and time formatting.

* **Collation**: It is a method for searching and sorting strings within a collection. It has a locale argument and is Unicode-aware.
* **Number Formatting**: Localized separators and digit grouping can be used to format numbers. Style formatting, numeral system, percent, and precision are among the other items.
* **Currency formatting**: Currency symbols, localized separators, and digit grouping are the most common ways to format numbers.
* **Date and time formatting**: Localized separators and ordering are used for formatting. The format can be short or long, and other characteristics such as location and time zone can be included

### **What are Proxy in ES6?**

The proxy objects are used to customize behaviour for basic operations like property lookup, assignment, enumeration, function invocation, etc.

We need to define three crucial terms:

* **handler** —  a placeholder object that holds the trap(s)
* **traps** — the method(s) that let you access a property.
* **target** — the virtualized object by the proxy

### **What is the difference between Set and WeakSet in ES6?**

| **Set** | **WeakSet** |
| --- | --- |
| A set can contain all types of values. | A weakSet can only contain objects. |
| Use .size to find the number of elements. | Use .length to find the number of elements. |
| .forEach() is available for iteration. | .forEach() is not available for iteration. |
| Nothing is auto-destroyed. | An element object will be auto released to the garbage collector if it has no other reference left. |

### **What is the advantage of using the arrow syntax for a constructor method?**

The main benefit of utilising an arrow function as a method within a constructor is that the value of ***this*** is set at the moment of function generation and cannot be changed later. As a result, whenever the constructor is used to create a new object, ***this*** refers to that object.

**Es6 features**

* The let keyword
* The const keyword
* Arrow Functions
* The ... Operator
* For/of
* Map Objects
* Set Objects
* Classes
* Promises
* Default Parameters
* Function Rest Parameter

**filter()**

Creates a new array with all of the elements of this array for which the provided filtering function returns true

**map()**

Creates a new array with the results of calling a provided function on every element in this array.

**reduce()**

Applies a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.

**What is generator function js?**

A generator is a process that can be paused and resumed and can yield multiple values. A generator in JavaScript consists of a generator function, which **returns an iterable Generator object**.

## What are Cookies?

Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem "how to remember information about the user":

* When a user visits a web page, his/her name can be stored in a cookie.
* Next time the user visits the page, the cookie "remembers" his/her name.

**Create cookie**

JavaScript can create, read, and delete cookies with the document.cookie property.

document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";

## Delete a Cookie with JavaScript

Deleting a cookie is very simple.

You don't have to specify a cookie value when you delete a cookie.

Just set the expires parameter to a past date:

document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";

**What id Dom ?**

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

DOM is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.

**What is mutable and immutable with an example in JavaScript?**

**Mutable can be changed or added to where immutable means something that cannot be changed or added**. Primitive values in JavaScript cannot have anything added upon to them, they can only be re-assigned, and hence all primitive values in JavaScript are immutable

**What is deep copy and shallow copy in JavaScript example?**

A deep copying means that value of the new variable is disconnected from the original variable while a shallow copy means that some values are still connected to the original variable.

let person = {

    firstName: 'John',

    lastName: 'Doe'

};

let copiedPerson = person;

Deepcopy ex

let person = {

    firstName: 'John',

    lastName: 'Doe',

    address: {

        street: 'North 1st street',

        city: 'San Jose',

        state: 'CA',

        country: 'USA'

    }

};

let copiedPerson = JSON.parse(JSON.stringify(person));

copiedPerson.firstName = 'Jane'; *// disconnected*

copiedPerson.address.street = 'Amphitheatre Parkway';

copiedPerson.address.city = 'Mountain View';

console.log(person);

**Localstorage and session storage**

**Event looping?**

JavaScript has a runtime model based on an event loop, which is **responsible for executing the code, collecting and processing events, and executing queued sub-tasks**.

**Foreach and map filter, reduce diff?**

**The forEach() method does not create a new array based on the given array**. The map() method creates an entirely new array. The forEach() method returns “undefined“. The map() method returns the newly created array according to the provided callback function.

**Foreach and for loop?**

**For Loops executes a block of code until an expression returns false while ForEach loop executed a block of code through the items in object collections**. For loop can execute with object collections or without any object collections while ForEach loop can execute with object collections only.