

INTERVIEW QUESTIONS ON JAVASCRIPT

1. What is java script?

Ans:

***Javascript is an object-based scripting language.**

***Which is lightweight and cross platform**

***Javascript is used to create interactive websites and JavaScript is used to create client-side dynamic pages.**

Ex:

```
<script>
    function addnumbers(a,b) {
        return a+b;
    }
    var sum = addnumbers(10,20)
    document. Write('sum of the numbers is ' + sum);

</script>
```

2. Why we use java script?

Ans:

It mainly used for:

***It is Client-side validation purpose**

***Dynamic drop-down menus**

***Displaying date and time**

***Displaying pop-up windows and dialog boxes**

***Displaying clocks etc**

***It can react to events**

***It can be used to validate data**

***it can be use to create cookies**

*It is designed with light weight features

*It is open source or cross platform

3. What are variables in java script?

Ans:

A javascript variables is simply a name of storage location.

A **variable** can store a single value.. JavaScript uses reserved keyword var to declare a **variable**. ... You can assign a value to a **variable** using equal to (=) operator when you declare it or before using it.

Ex:

```
var Number = 5;  
var String = "Hi!";  
var boolen1 = true;
```

2 types of variables in javascript

Local variable,global variable

1. Local variable:

A javascript local variable is declared inside the block or function.

It is accessible within the function or block only.

```
<!DOCTYPE html>  
<html lang="en-US">  
<head>  
  <meta charset="UTF-8">  
  <title>js local var</title>  
</head>  
  
<body>  
  
<script type="text/javascript">  
  
function local(){  
  var x = 10; //local variable  
}
```

```
</script>
</body>
</html>
```

2. global variable

A Javascript global variable is accessible from any function. A variable declared outside the function

Or declared with window object is known as global variable.

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>js local var</title>
</head>

<body>

<script type="text/javascript">
var data=200;
function global1(){
  document.writeln(data) //global variable
}

function global2(){
  document.writeln(data)
}

global1(); //function calling
global2();

</script>
</body>
</html>
```

4. What is function in java script?

Ans:

Functions:

A **JavaScript function** is a block of code designed to perform a particular task.

*Javascript functions are used to perform operations.

*we can call javascript function many times to reuse the code.

A **JavaScript function** is executed when "something" invokes it (calls it).

...

Function Invocation

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

-When it is invoked (called) from **JavaScript** code.

-Automatically (self invoked)

Ex:

```
function myFunction() {  
    alert("ALERT!");  
}
```

```
myFunction();
```

Advantages:

1.code reusability: we can call a function several times so it save coding.

2.Less coding: We don't need to write many lines of code Each time to perform a common task.

1.Calling Function:

*TO invoke a function somewhere later in the script.

```
<!DOCTYPE html>  
<html lang="en-US">  
<head>  
    <meta charset="UTF-8">  
    <title>Calling Function</title>  
</head>  
  
<body>  
  
<p>click the following button</p>
```

```
<form>
  <input type="button" value="sayhello" onclick="sayhello()">
</form>

<script>
//demo
// function sayhello(){
// alert("hiiii")
// }

function sayhello(){
document.write("hello world")
}

function(){
console.log('lorem ipsum')

};

</script>
</body>
</html>
```

2.Function Parameters

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>Function with parameters</title>
</head>

<body>

<p>click the following button</p>

<form>
```

```

    <input type="button" value="sayhello" onclick="sayhello('praveen',25)">
</form>

<script>

function sayhello(name,age){
document.write(name + "is" + age + "years old");
}

</script>
</body>
</html>

```

3.Function with return type

```

<!DOCTYPE html>
<html lang="en-US">
<head>
    <meta charset="UTF-8">
    <title>Function with return type</title>
</head>

<body>

<script>

function sayhello(name,age){
return "hii this is return type";
}

document.write(sayhello())

</script>
</body>
</html>

```

5. What are array methods in java script?

Ans:

The Array object lets you store multiple values in a single variable it stores a fixed size sequential collection of elements of the same type.

Concat,copyWithin,entries

every

fill

filter

find

findIndex

for each

from

includes

indexOf

isArray

keys

lastIndexOf

map

pop

push

reduce

reduce right

join

reverse

shift

unshift

slice

splice

some

sort

toString

valueOf

6. What is push(), pop(), shift(), unshift(), index of(), includes(), concat(), for Each(), every(), some(), map(), reduce() methods in java script?

Ans:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

1.Create an Array

```
let fruits = ['Apple', 'Banana']  
  
console.log(fruits.length)
```

```
// 2
```

2. Array.prototype.push()

Adds one or more elements to the end of an array, and returns the new **length** of the array.

```
const fruits = []
fruits.push('banana', 'apple', 'peach')

console.log(fruits.length) // 3
```

3. Array.prototype.pop()

Removes the last element from an array and returns that element.

```
var myFish = ['angel', 'clown', 'mandarin', 'sturgeon'];

var popped = myFish.pop();

console.log(myFish); // ['angel', 'clown', 'mandarin' ]

console.log(popped); // 'sturgeon'
```

4. Array.shift()

The **shift()** method removes the **first** element from an array and returns that removed element. This method changes the length of the array.

```
const array1 = [1, 2, 3];

const firstElement = array1.shift();

console.log(array1);
// expected output: Array [2, 3]

console.log(firstElement);
// expected output: 1
```

5. Array.unshift()

The **unshift()** method adds one or more elements to the beginning of an array and returns the new length of the array.


```
const array1 = [1, 2, 3];

console.log(array1.unshift(4, 5));
// expected output: 5

console.log(array1);
// expected output: Array [4, 5, 1, 2, 3]
```

6. Array.indexOf()

Find the index of an item in the Array

The `indexOf()` method returns the first index at which a given element can be found in the array, or -1 if it is not present.

```
const beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];

console.log(beasts.indexOf('bison'));
// expected output: 1

// start from index 2
console.log(beasts.indexOf('bison', 2));
// expected output: 4

console.log(beasts.indexOf('giraffe'));
// expected output: -1
```

7. Array.includes()

The `includes()` method determines whether an array includes a certain value among its entries, returning `true` or `false` as appropriate.

```
const array1 = [1, 2, 3];

console.log(array1.includes(2));
// expected output: true

const pets = ['cat', 'dog', 'bat'];

console.log(pets.includes('cat'));
// expected output: true

console.log(pets.includes('at'));
// expected output: false
```

8. Array.concat()

The `concat()` method is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

```
const array1 = ['a', 'b', 'c'];
const array2 = ['d', 'e', 'f'];
const array3 = array1.concat(array2);

console.log(array3);
// expected output: Array ["a", "b", "c", "d", "e", "f"]
```

9. Array.forEach()

The `forEach()` method executes a provided function once for each array element.

```
const array1 = ['a', 'b', 'c'];

array1.forEach(element => console.log(element));

// expected output: "a"
// expected output: "b"
// expected output: "c"
```

```
var sum = 0;
var arr = [10,18,12,20];

arr.forEach(function myFunction(element) {
    sum= sum+element;
    document.writeln(sum);
});
```

o/p: 10 28 40 60

The JavaScript array `forEach()` method is used to “invoke the specified function once for each array element”.

10. Array.every()

Returns `true` if every element in this array satisfies the testing function.

The `every()` method tests whether all elements in the array pass the test implemented by the provided function. It returns a Boolean value.

```
const isBelowThreshold = (currentValue) => currentValue < 40;
```

```
const array1 = [1, 30, 39, 29, 10, 13];

console.log(array1.every(isBelowThreshold));
// expected output: true
```

11. Array.some()

Returns `true` if at least one element in this array satisfies the provided testing function.

The `some()` method tests whether at least one element in the array passes the test implemented by the provided function. It returns a Boolean value.

```
const array = [1, 2, 3, 4, 5];

// checks whether an element is even
const even = (element) => element % 2 === 0;

console.log(array.some(even));
// expected output: true
```

12. Array.map()

Returns a new array containing the results of calling a function on every element in this array.

The `map()` method **creates a new array** populated with the results of calling a provided function on every element in the calling array.

```
const array1 = [1, 4, 9, 16];

// pass a function to map
const map1 = array1.map(x => x * 2);

console.log(map1);
// expected output: Array [2, 8, 18, 32]
```

13. Array.reduce()

Apply a function against an accumulator and each value of the array (from left-to-right) as to reduce it to a single value.

The `reduce()` method executes a **reducer** function (that you provide) on each element of the array, resulting in single output value.

```
const array1 = [1, 2, 3, 4];
const reducer = (accumulator, currentValue) => accumulator + currentValue;
```

```
// 1 + 2 + 3 + 4
console.log(array1.reduce(reducer));
// expected output: 10

// 5 + 1 + 2 + 3 + 4
console.log(array1.reduce(reducer, 5));
// expected output: 15
```

7. What are string methods?

Ans:

1. `charAt()` It returns the character of the given index.

```
const sentence = 'The quick brown fox jumps over the lazy dog.';

const index = 4;

console.log(`The character at index ${index} is ${sentence.charAt(index)}`);
// expected output: "The character at index 4 is q"
```

2. `concat()` It returns the combined result of two or more string.

```
const str1 = 'Hello';
const str2 = 'World';

console.log(str1.concat(' ', str2));
// expected output: "Hello World"

console.log(str2.concat(', ', str1));
// expected output: "World, Hello"
```

3. `endsWith()` It is used to check whether a string ends with another string.

The `endsWith()` method determines whether a string ends with the characters of a specified string, returning `true` or `false` as appropriate.

```
const str1 = 'Cats are the best!';

console.log(str1.endsWith('best', 17));
// expected output: true
```

```
const str2 = 'Is this a question';

console.log(str2.endsWith('?'));
// expected output: false
```

4. `includes()` It checks whether the string contains another string or not.

The `includes()` method determines whether one string may be found within another string, returning `true` or `false` as appropriate.

```
const sentence = 'The quick brown fox jumps over the lazy dog.';

const word = 'fox';

console.log(`The word "${word}" ${sentence.includes(word) ? 'is' : 'is not'} in the sentence`);
// expected output: "The word "fox" is in the sentence"
```

5. `indexOf()` It returns the index of the first occurrence of the specified substring from a string, otherwise returns -1.

```
const paragraph = 'The quick brown fox jumps over the lazy dog. If the dog barked, was it really lazy?';

const searchTerm = 'dog';
const indexOfFirst = paragraph.indexOf(searchTerm);

console.log(`The index of the first "${searchTerm}" from the beginning is ${indexOfFirst}`);
// expected output: "The index of the first "dog" from the beginning is 40"

console.log(`The index of the 2nd "${searchTerm}" is ${paragraph.indexOf(searchTerm, (indexOfFirst + 1))}`);
// expected output: "The index of the 2nd "dog" is 52"
```

6. `lastIndexOf()` It returns the index of the last occurrence of a value in the string.

```
const paragraph = 'The quick brown fox jumps over the lazy dog. If the dog barked, was it really lazy?';
```

```
const searchTerm = 'dog';

console.log(`The index of the first "${searchTerm}" from the end is ${paragraph.lastIndexOf(searchTerm)}`);
// expected output: "The index of the first "dog" from the end is 52"
```

7. `match()` It is used to match a regular expression against the given string.

```
const paragraph = 'The quick brown fox jumps over the lazy dog. It barked.';
const regex = /[A-Z]/g;
const found = paragraph.match(regex);

console.log(found);
// expected output: Array ["T", "I"]
```

8. `replace()` It replaces the matched substring with the new substring.

```
const p = 'The quick brown fox jumps over the lazy dog. If the dog reacted, was it really lazy?';

console.log(p.replace('dog', 'monkey'));
// expected output: "The quick brown fox jumps over the lazy monkey. If the dog reacted, was it really lazy?"

const regex = /Dog/i;
console.log(p.replace(regex, 'ferret'));
// expected output: "The quick brown fox jumps over the lazy ferret. If the dog reacted, was it really lazy?"
```

9. `search()` It searches for a match between a regular expression and string.

```
const paragraph = 'The quick brown fox jumps over the lazy dog. If the dog barked, was it really lazy?';

// any character that is not a word character or whitespace
const regex = /^[^\w\s]/g;

console.log(paragraph.search(regex));
// expected output: 43
```

```
console.log(paragraph[paragraph.search(regex)]);  
// expected output: "
```

10. `slice()` It returns a section of a string.

```
const str = 'The quick brown fox jumps over the lazy dog.';  
  
console.log(str.slice(31));  
// expected output: "the lazy dog."  
  
console.log(str.slice(4, 19));  
// expected output: "quick brown fox"  
  
console.log(str.slice(-4));  
// expected output: "dog."  
  
console.log(str.slice(-9, -5));  
// expected output: "lazy"
```

11. `split()` It splits the string into substrings and returns an array.

```
const str = 'The quick brown fox jumps over the lazy dog.';  
  
const words = str.split(' ');  
console.log(words[3]);  
// expected output: "fox"  
  
const chars = str.split('');  
console.log(chars[8]);  
// expected output: "k"  
  
const strCopy = str.split();  
console.log(strCopy);  
// expected output: Array ["The quick brown fox jumps over the lazy dog."]
```

12. `substring()` It returns a string between the two given indexes.

The `substring()` method returns the part of the `string` between the start and end indexes, or to the end of the string.

```
const str = 'Mozilla';
```

```
console.log(str.substring(1, 3));  
// expected output: "oz"  
  
console.log(str.substring(2));  
// expected output: "zilla"
```

The **substr()** method returns a portion of the string, starting at the specified index and extending for a given number of characters afterwards.

```
const str = 'Mozilla';  
  
console.log(str.substr(1, 2));  
// expected output: "oz"  
  
console.log(str.substr(2));  
// expected output: "zilla"
```

13. toLowerCase() It converts the all characters of a string into lower case.

```
const sentence = 'The quick brown fox jumps over the lazy dog.';  
  
console.log(sentence.toLowerCase());  
// expected output: "the quick brown fox jumps over the lazy dog."
```

14. toUpperCase() It converts the all characters of a string into upper case.

```
const sentence = 'The quick brown fox jumps over the lazy dog.';  
  
console.log(sentence.toUpperCase());  
// expected output: "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG."
```

15. trim() It is used to trims the white space from the beginning and end
of the string.

```
const greeting = '    Hello world!    ';  
  
console.log(greeting);  
// expected output: "    Hello world!    ";  
  
console.log(greeting.trim());  
// expected output: "Hello world!";
```


16. `trimLeft()` or `trimStart()` It is used to trim the white space from the left side of the string.

```
const greeting = '  Hello world!  ';

console.log(greeting);
// expected output: "  Hello world!  ";

console.log(greeting.trimStart());
// expected output: "Hello world!  ";
```

17. `trimRight()` or `trimEnd()` It is used to trim the white space from the right side of the string.

```
const greeting = '  Hello world!  ';

console.log(greeting);
// expected output: "  Hello world!  ";

console.log(greeting.trimEnd());
// expected output: "  Hello world!";
```

18. `valueOf()` It returns a primitive value of the specified object.

```
const stringObj = new String('foo');

console.log(stringObj);
// expected output: String { "foo" }

console.log(stringObj.valueOf());
// expected output: "foo"
```

8. How can we apply style in java script?

Ex:

```
<html>
  <body>

    <p id="p2">Hello World!</p>
```

```

<script>
document.getElementById("p2").style.color = "blue";
</script>

<p>The paragraph above was changed by a script.</p>

</body>
</html>
Ex:2
// Here are two different ways
document.getElementById("myH1").style.color = "red";

element.style.backgroundColor = "red"; // set the background color of an element to red

```

9. What are loops in java script?

Ans:

Defenite Loops

JavaScript loops are used to repeatedly run a block of code - until a certain condition is met

Ex:

```

for(var i = 1; i<=5; i++){
document.write(i + "<br/>")
}

```

* in this loop, we know about the number of iterations before the execution of the block of statements.

1.for loop

* A loop is a repetition control structure. It is used to execute the block of code to a specific number of times.

*A for statement contains the initialization, condition and increment and decrement in a single line.

Syntax:

```

for (first expression; second expression; third expression){
//statements to be executed repeatedly
}

```

Initialization: it allows us to declare and initialize the loop control variables

Condition: if it is true, the loop gets executed. If it is false , the loop does not executed.

Increment/decrement: it allows us to update the loop control variables.

1.

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>js for</title>

</head>

<body>

<script type="text/javascript">

for(var i = 1; i<=5; i++){
document.write(i + "<br/>")
}

</script>
</body>
</html>
```

o/p:

1

2

3

4

5

2.while loop

* while loop iterates the elements for the infinite number of times.

* it executes for the infinite number times

* it executes the instruction repeatedly until the specific condition is true.

* we can use it when the number of iteration is not known.

Syntax:

```
While(condition){  
    //code to be executed  
}
```

```
<!DOCTYPE html>  
<html lang="en-US">  
<head>  
    <meta charset="UTF-8">  
    <title>js for</title>  
  
</head>  
  
<body>  
  
<script type="text/javascript">  
  
var i = 11;  
  
while(i<=15){  
document.write(i + "<br/>")  
i++;  
}  
  
</script>  
</body>  
</html>
```

o/p:

11

12

13

14

15

Explanation of while loop:

- 1.while loop start the execution with checking the condition.
- 2.if the condition true .the loop body statement gets executed.
3. otherwise, the first statement following the loop gets executed
- 4.if condition false loop gets terminated.

3.do-while-loop

* do-while loop iterates the elements for the infinite number of times similar to the while loop.

It gets executed at least once whether the condition is true.

* the when the no.of iterations are not fixed in do-while loop.

* you have to execute the loop at least once.

Syntax:

```
Do{  
    //code to be executed  
}while(condition);
```

```
<!DOCTYPE html>  
<html lang="en-US">  
<head>  
    <meta charset="UTF-8">  
    <title>js for</title>  
</head>  
<body>  
  
<script type="text/javascript">  
  
var i = 11;  
  
do{  
document.write(i + "<br/>")  
i++;  
}  
while(i<=15);  
  
</script>
```

```
</body>
</html>
```

o/p:

11

12

13

14

15

Explanation of do-while:

1.do-while starts executing the statement without checking any condition for the first time.

2. after the execution of the statement and update of the variable.

3.if condition is true, the next iteration of the loop start execution

4.if the condition false loop gets terminated

If false

```
let n = 10;

do{
  console.log(n);
  n++;
}
while(n<=5)
```

o/p:

```
E:\1 NOV DEC 2019\angular app\22indefiniteloops>tsc dowhile.ts
E:\1 NOV DEC 2019\angular app\22indefiniteloops>node dowhile.ts
10
```

3. for ..in:

*** The loop is used with an array,list or tuple. This loop iterates through a list or collection and return an index on each iteration.**

* the datatype of “val” should be a string or any

Syntax:

```
For(let val in list){  
  //statements  
}
```

10. What is event loop in java script?

Ans:

The **event loop** continuously checks the call stack to see if there's any function that needs to run. While doing so, it adds any function call it finds to the call stack and executes each one in order.

11. What is debugging?

Ans:

The **debugger** statement stops the execution of **JavaScript**, and calls (if available) the **debugging** function. Using the **debugger** statement has the same function as setting a breakpoint in the code.

12. What is event propagation?

Ans: **Event propagation** is a way to describe the “stack” of **events** that are fired in a web browser.

The standard DOM Events describes 3 phases of event propagation:

- Capturing phase – the event goes down to the element.
- **Target** phase – the event reached the **target** element.
- Bubbling phase – the event bubbles up from the element.

13. What is event delegation?

Ans: **JavaScript** event **delegation** is a simple technique by which you add a single event handler to a parent element in order to avoid having to add event handlers to multiple child elements

14. What is event bubbling?

Ans: **Event bubbling** is a type of **event** propagation where the **event** first triggers on the innermost target element, and then successively triggers on the ancestors (parents) of the target element in the same

nesting hierarchy till it reaches the outermost DOM element or document object (Provided the handler is initialized)

when the lower hierarchy event calling .. rest of all events are calling one by one..

The process is called “bubbling”, because events “bubble” from the inner element up through parents like a bubble in the water.

Ex:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width">
<title>Event Bubbling</title>
</head>
<body>
<div id="form">
  <div id="parent">
    <button id="child">Child</button>
  </div>
</div>
<script>

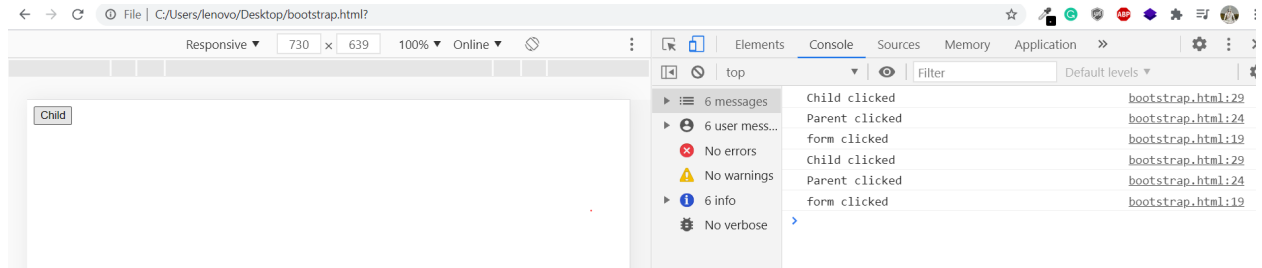
var form = document.querySelector('#form');
// <!-- Add click event on form -->
form.addEventListener('click', function(){
console.log("form clicked");
});
var parent = document.querySelector('#parent');
// <!-- Add click event on parent div -->
parent.addEventListener('click', function(){
console.log("Parent clicked");
});
var child = document.querySelector('#child');
// <!-- Add click event on child button -->
child.addEventListener('click', function(){
console.log("Child clicked");
});
```



```

</script>
</body>
</html>

```



```

<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Javascript Programs</title>
    <!-- css -->
    <!-- <link rel="stylesheet" href="form.css"> -->
    <style>
      div,p,form{
        border: 2px solid red;
        margin: 40px;
      }
    </style>

  </head>

  <body>

    <!-- javascript code starts here -->

    <form onclick="alert('this is form')">I am Form
      <div onclick="alert('div')">I am div
        <p onclick="alert('para')">I am Para</p>
      </div>
    </form>

```

```
<script>

</script>

<!-- javascript code ends here -->

<!-- <script type="text/javascript" src="form.js"></script> -->
</body>

</html>
```

I am Form

I am div

I am Para

15. What is event capturing?

Ans: **Event capturing** is the **event** starts from top element to the target element. It is the opposite of **Event bubbling**, which starts from target element to the top element.

16. What is event throttling?

Ans: **Throttling** is a technique in which, no matter how many times the user fires the **event**, the attached function will be executed only once in a given time interval. For instance, when a user clicks on a button, a `helloWorld` function is executed which prints Hello, world on the console.

Ex:

```
function throttle (callback, limit) {
  var wait = false; // Initially, we're not waiting
  return function () { // We return a throttled function
    if (!wait) { // If we're not waiting
      callback.call(); // Execute users function
      wait = true; // Prevent future invocations
      setTimeout(function () { // After a period of time
        wait = false; // And allow future invocations
      }, limit);
    }
  }
}

// Usage Example:
// On scroll, allow function to run at most 1 time per 100ms
window.addEventListener("scroll", throttle(function(){
  /*stuff to be throttled*/
}, 100));
```

17. What is name space pollution in java script?

Ans: **Namespace** refers to the programming paradigm of providing scope to the identifiers (names of types, functions, variables, etc) to prevent collisions between them. For instance, the same variable name might be required in a program in different contexts.

18. What is enumerable properties in java script?

Ans: An **enumerable** property in **JavaScript** means that a property can be viewed if it is iterated using the `for...in` loop or `Object...` To explicitly change the internal **enumerable** attribute of a property, the `Object`.

19. What is non-numeric object properties in java script?

Ans:

20. What is the behaviour of this keyword?

Ans: The **this** keyword in JavaScript is very important concept but at the same time very confusing to understand.

In JavaScript, **this** keyword refers to the object it belongs to. It has different values depending on where it is used:

- In a method, **this** refers to the **owner object** where method is defined.
- Alone, **this** refers to the **global object**.

- In a function, `this` refers to the **global object**.
- In a function, in strict mode, `this` is undefined.
- When a function called with **new** keyword, `this` refers to **new object instance**
- In a DOM event, `this` refers to the **element** that received the event.
- Function prototype methods `call()`, `apply()` and `bind()` can be used to refer `this` to **any object**.

Anss

```
function fun(){
  "use strict"
  console.log(this)
}
fun();
```

21. What is call, apply and bind write syntax?

Ans:

The `bind()` method creates a new function that, when called, has its `this` keyword set to the provided value,

The `bind()` method returns a new [function](#), when invoked, has its `this` sets to a specific value.

```
var car = {
  registrationNumber: "GA12345",
  brand: "Toyota",

  displayDetails: function(ownerName){
    console.log(ownerName + ", this is your car: " + this.registrationNumber
+ " " + this.brand);
  }
}
var myCarDetails = car.displayDetails.bind(car, "Vivian"); // Vivian, this is your car: GA12345 Toyota
myCarDetails();
```

CALL Method Example:

The `call()` method is a predefined JavaScript method.

It can be used to invoke (call) a method with an owner object as an argument (parameter).

```
var person1 = {  
  firstName:"John",  
  lastName: "Doe"  
}  
  
var person2 = {  
  firstName:"Mary",  
  lastName: "Doe"  
}  
  
var person = {  
  fullName: function(p1,p2) {  
    return this.firstName + " " + this.lastName + " " + p1 + p2;  
  }  
}  
  
person.fullName.call(person1, "ECE", "abc"); // Will return "John Doe"  
console.log(person.fullName.call(person1, "ECE", "abc"))
```

Apply Methods Example:

```
var person1 = {
```

```

firstName:"John",

lastName: "Doe"

}

var person2 = {

firstName:"Mary",

lastName: "Doe"

}

var person = {

fullName: function(p1,p2,p3) {

return this.firstName + " " + this.lastName + " " + p1 + p2 + p3;

}

}

//person.fullName.call(person1, "ECE", "abc"); // Will return "John Doe"

console.log(person.fullName.apply(person1, ["ECE", "abc", "p3"]))

```

The difference is:

The `call()` method takes arguments **separately**.

The `apply()` method takes arguments as an **array**.

Bind Method:

-The **bind()** method creates a new **function**, when invoked, has the `this` sets to the provided value.

-The **bind()** method allows an object to borrow a **method** from another object without making a copy of that **method**. This is known as **function borrowing** in **JavaScript**.

```
func.bind(this)
```

22. What is closure?

Ans: *Closure means that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.*

You have learned that we can create [nested functions](#) in JavaScript. Inner function can access variables and parameters of an outer function (however, cannot access arguments object of outer function). Consider the following example.

```
function OuterFunction() {  
    var outerVariable = 1;  
  
    function InnerFunction() {  
        alert(outerVariable);  
    }  
  
    InnerFunction();  
}
```

In the above example, InnerFunction() can access outerVariable.

Now, as per the definition above, InnerFunction() can access outerVariable even if it will be executed separately. Consider the following example.

Example: Closure

```
function OuterFunction () {  
  
var outerVariable = 100;  
  
function InnerFunction() {  
    console.log(outerVariable);  
}  
  
return InnerFunction;  
}  
var innerFunc = OuterFunction();  
  
innerFunc(); // 100
```

23. Give some real time examples of closure?

24. Have you done any project on java script?

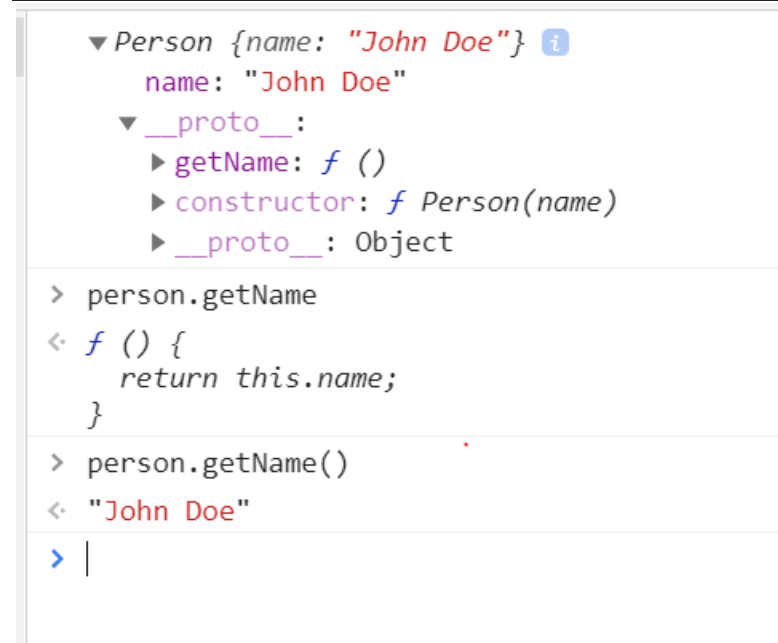
25. I don't know the height and width of the div and align that div at center. how can we do that in java script?

26. What is prototype in java script?

Ans: The **prototype** is an object that is associated with every functions and objects by default in **JavaScript**, where function's **prototype** property is accessible and modifiable and object's **prototype** property (aka attribute) is not visible. ... Every function includes **prototype** object by default.

Example:

```
function Person(name) {  
  this.name = name;  
}  
Person.prototype.getName = function() {  
  return this.name;  
}  
  
var person = new Person("John Doe");  
person.getName() //"John Doe"  
  
console.log(person)
```



27. What is hoisting ?

Ans: 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.

Hoisting in Javascript is when Javascript moves variable declarations (NOT definitions) up to the top of its global or local scope. This means that var , const , and let variable declarations are interpreted as if it is at the top of its scope

28. How can we create object in java script ?

Ans: There are four ways to create an object in JavaScript - using object literals, using the function constructor, using the **Object.create** method, and using the class keyword

1. By object literal

```
emp= {id:23,name:"praveen",salary:"40000"};
emp2= {id:23,name:"kumar",salary:"50000"};
emp3= {id:24,name:"mahendra",salary:"60000"};
document.write(emp.id+ " " + emp.name+ " "+ emp.salary)
document.write(emp2.id+ " " + emp2.name+ " "+ emp2.salary)
alert(emp3.id+ " " + emp3.name+ " "+ emp3.salary)
```

2. By creating instance of Object directly (using new using keyword)

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>Instance of object </title>
</head>

<body>

<script>

var emp= new Object ();
emp.id=23;
emp.name="alex";
emp.salary = 50000;

document.write (emp.id+ " " + emp.name+ " "+ emp.salary)

var emp= new Object ();
emp.id=254;
emp.name="alexascas";
emp.salary=500000;
document.write(emp.id+ " " + emp.name+ " "+ emp.salary)

</script>
```

```
</body>
</html>
```

3. 3.By using an object constructor (using new keyword)

*you need to create function with arguments.

*Each argument value can be assigned In the current object by using this keyword.

*the this keyword refers to the current object.

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <meta charset="UTF-8">
  <title>Object Constructor</title>
</head>

<body>

<script>

// function emp(){
// alert("hiii")
// }
// emp();

function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;
}
e = new emp(10,"praveen",10000);
document.write(e.id+" "+e.name+" "+e.salary)
</script>
</body>
</html>
```

4.using the Object.Create() method

```
var Car1 = {
  model:'BMW',
  color:'red'
}
```

```
var ElectricCar = Object.create(Car1,{ym:{
  writable:true,
  enumerable: true,
  configuration:false,
  value:1984
}});
document.write(ElectricCar.model)
console.log(ElectricCar.model)
```

29. Create a construction object and pass an argument ?

Ans:

30. How can define class in java script using custom construction?

Ans:

```
function Car (make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
}

const car1 = new Car('Eagle', 'Talon TSi', 1993);

console.log(car1.make);
// expected output: "Eagle"
```

31. What is primitive data type and non-primitive data type ?

Ans:

Non-primitive types are created by the programmer and is not defined by Javascript.

Primitive types are predefined (already defined) in Javascript.

- In JavaScript, a primitive (primitive **value**, primitive data type) is data that is not an object and has no methods.

- There are 6 primitive data types: **string**, number, **Boolean**, **undefined**, and null,any,void

- Non-**Primitive data types** refer to objects and hence they are called reference **types**. Examples of non-**primitive types** include Strings, Arrays, objects, Classes, Interface, etc.

32. Why primitive data type called as primitive data type and non-primitive data type called as non-primitive data type?

Ans:

33. If you declare variable with let and const keyword will hoisting work on root ?

Ans:

Hoisting is **JS's** default behavior of defining all the declarations at the top of the scope before code execution. One of the benefits of **hoisting** is that it enables us to call functions before they appear in the code. **JavaScript** only hoists declarations, not initializations.

Hoisting in Javascript is when Javascript moves variable declarations (NOT definitions) up to the top of its global or local scope. This means that **var** , **const** , and **let** variable declarations are interpreted as if it is at the top of its scope

Ans....

Because the declaration and initialization phases are decoupled, **hoisting** is **not** valid for a **let** variable (including for **const** and class). Before initialization, the variable is in temporal dead zone and is **not** accessible

34. Difference between promise and observable?

Ans:

=> PROMISES : promises are worked with asynchronous operations and it returns only one value either promise-resolve or promise-reject.

=> OBSERVABLES : It means A sequence of data that are arriving asynchronously from the server over a time with help of subscribe method.

here we use http-cal and http-response.

import {Observable} from 'rxjs package.

35. A=30

Documents write [a]

Var a

```
a=30
document.write(a)
var a;
```

This syntax is wrong or right ?

Ans: Yes it is right becoz var is global scope

36 how to remove duplicate element from an array ?

A = [20, 30, 40, 50, 40]

Result should be ; [20, 30, 40, 50,]

A [Set](#) is a collection of unique values. To remove duplicates from an [array](#):

- First, convert an array of duplicates to a **Set**. The new **Set** will implicitly remove duplicate elements.
- Then, convert the **set** back to an array.
- Let A = [20, 30, 40, 50, 40]

Let unique = [... new set(A)];

Console.log(unique);

2) Remove duplicates from an array using indexOf() and filter() methods

The [indexOf\(\)](#) method returns the index of the first occurrence of an element in an array. For example:

37 var x = [2, 3, 2, 4, 3, 2]

Result : [4, 3, 2, 2, 2]

Write a logic ?

```
var x = [2, 3, 2, 4, 3, 2 ]  
  
console.log(x.sort().reverse())
```

38 var x= "java script is net java" ;

Var y="java" ;

Count string occurrence?

```
var x= "java script is net java" ;  
var y="java";  
  
let count = x.match(/java/g);  
  
console.log(count); //2  
  
let count2 = y.match(/a/g);  
  
console.log(count2); //2
```

39 function xyz(){

Console.log(x)

Var x = 10;

}

Output?

```
function xyz(){  
console.log(x);  
var x = 10;  
}
```

No output.empty but no errors

40 (function () {

 Console.log [1];

 Set time out (function () console.log (2) ;1000);

 Set time out (function () console.log (3) ,0);

 Console.log (4);

})();

```
( function () {  
console.log(1) ;  
setTimeout( function () {console.log (2)} ,1000);  
setTimeout( function () {console.log (3)} ,0);  
console.log (4);  
})(); // 1 4 3 2
```

O/p: _?

41 take two function both function have for loop

(I) For (I=0 ; ; <= 1000 ; I + +) {

 Alert("hii")

}

(II) for (=0 ; I <= 1000 ;I ++) {

 Alert ('hi there ') ;

}

Guess; how they will execute

 In synchronous way ,or one by one on

 Function (I) will execute then function (ii) will

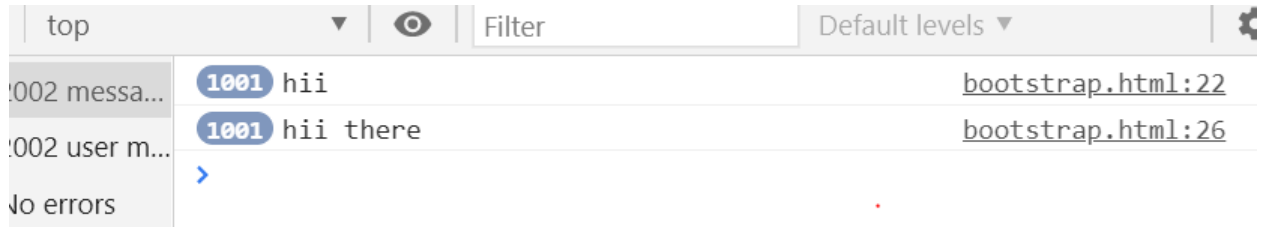
Execute ?

Ans:

```
for(i=0;i<=1000;i++){  
    console.log("hii")  
}
```

```
for(i=0;i<=1000;i++){
  console.log("hii there")
}
```

synchronous way



42 how can we get that of properties?

43 do you know MIME?

Ans:

-text/javascript. Per the HTML specification, **JavaScript** files should always be served using the **MIME type text/javascript**

- **MIME types**—also sometimes called Internet media **types** or Content-**types**—describe the media **type** of content either contained in email or served by web servers or web applications, and are intended to help guide a web browser to correctly process and display the content.

44 What is phase production?

45 what is IIFE?(Immediately invoked function expression ?

Ans:

An **Immediately-invoked Function Expression (IIFE for friends)** is a way to execute functions immediately, as soon as they are created.

IIFEs are very useful because they don't pollute the global object, and they are a simple way to isolate variables declarations..

these functions are stored in window object..

This functions wont gets called until we make explicit call.

These functions will be stored in globally, we can re-use anywhere.(window).

but, IIFE funtions are not stored in window object


```
//IIFE or Funtion Expression

(function(){
    console.log("I am IIFE Function");
})
```

```
// iife with parameters
(
    function(a){
        console.log("this is IIFE function" + a);
    }
)(12)
```

Example:

```
result = (function(a, b){
    return a - b;
})(100, 42);

console.log(result); // 58
```

46 what will be output?

(i) For(var l=0;l<3;l++) (II) for(let l=0;l<3;l++)

```
{
    {
        If(l<3)        if(l<3)
        Console.log(l)    console.log(l)
    }
}
```

```
for(var i=0;i<3;i++) {
    if(i<3)
        console.log(i)    //0,1,2
}
```

```
for(let i=0;i<3;i++) {
    if(i<3)
        console.log(i)    //0,1,2
}
```

47 have you used any third party library like chartered library on rich text library on any other?

48 what is cookies?

Ans:

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. ... When a user visits a web page, his/her name can be stored in a **cookie**.

49 what is default time out for cookies?

50 difference between cookies and session storage?

Ans:

Cookies and Sessions are used to store information. **Cookies** are only stored on the client-side machine, while **sessions** get stored on the client as well as a server. A **session** creates a file in a temporary directory on the server where registered **session** variables and their values are stored.

: Cookies i) maximum data size 4kb ii) support string data type iii) server side accessible support iv)data transferred on every http request v) can be accessed server side and client side vi) life time as specified vii) no secure data storage

LocalStorage: I) maximum data size 5mb ii) support string data type iii) server side not accessible iv)No data transferred on every http request v) Accessed client side vi) life time till deleted vii) no secure data storage

Session Storage: I) maximum data size 5mb ii) support string data type iii) server side not accessible iv)No data transferred on every http request v) Accessed client side vi) life time till tab or window is closed vii) no secure data storage

51 try one call is dependent on another call if will breakdown the java script code ...right

How does java script control that?

52 difference between var and let?

Ans: The main **difference between let and var** is that scope of a variable defined with **let** is limited to the block in which it is declared while variable declared with **var** has the global scope. So we can say that **var** is rather a keyword which defines a variable globally regardless of block scope

Example:

```
function run() {  
  var foo = "Foo";  
  let bar = "Bar";  
  
  console.log(foo, bar);  
  
  {  
    let baz = "Bazz";  
    console.log(baz);  
  }  
}
```

var declarations are globally scoped or function scoped while let and const are block scoped.

var variables can be updated and re-declared within its scope; let variables can be updated but not re-declared; const variables can neither be updated nor re-declared.

They are all hoisted to the top of their scope. But while var variables are initialized with undefined, let and const variables are not initialized.

While var and let can be declared without being initialized, const must be initialized during declaration.

- 1). var and let can change their value and const cannot change its value
- 2). var can be accessible anywhere in function but let and const can only be accessible inside the block where they are declared.
- 3). let and const have a block scope but var has function scope.

```
//var:  
function nodeSimplified(){  
  var a =10;  
  console.log(a); // output 10  
  if(true){  
    var a=20;  
    console.log(a); // output 20  
  }  
  console.log(a); // output 20  
}  
  
//let:  
function nodeSimplified(){
```

```

let a =10;
console.log(a); // output 10
if(true){
  let a=20;
  console.log(a); // output 20
}
console.log(a); // output 10
}

//const:
function nodeSimplified(){
  const MY_VARIABLE =10;
  console.log(MY_VARIABLE); //output 10
  MY_VARIABLE =20; //throws type error
  console.log(MY_VARIABLE);
}

```

53 difference between == and ===?

Ans:

== in **JavaScript** is used for comparing two variables, but it ignores the datatype of variable.

=== is used for comparing two variables, but this operator also checks datatype and compares two values. Checks the equality of two operands without considering their type.

Example:

```

var one = 1;
var one_again = 1;
var one_string = "1"; // note: this is string

console.log(one == one_again); // true
console.log(one === one_again); // true
console.log(one == one_string); // true.
console.log(one === one_string); // false.

```

54 create one object and write syntax for this?

Ex1:

```

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

```

Ex2:

```

        let obj = {
// fields
    name:"value",
    num: 123,
//methods
    foo: function(){
        return "hello"
    }
}

console.log(obj.foo()) //hello
console.log(obj.name) //value

```

55 what is promise?

Ans:

PROMISES: promises are worked with asynchronous operations and it returns only one value either promise-resolve or promise-reject.

A **promise** is an object that may produce a single value..

A **promise** may be in one of 3 possible states: fulfilled, rejected, or pending

"Producing code" is code that can take some time

"Consuming code" is code that must wait for the result

A Promise is a JavaScript object that links producing code and consuming code

Example:

```

let myPromise = new Promise(function(myResolve, myReject) {
    let x = 0;

// some code (try to change x to 5)

    if (x == 0) {
        myResolve("OK");
    } else {
        myReject("Error");
    }
});

```

```
myPromise.then();
console.log(myPromise)
Promise {<fulfilled>: "OK"}
```

56 what is ASYNC await?

Ans:

Await:

Await function is used to wait for the promise. It could be used within the async block only. It makes the code wait until the promise returns a result. It only makes the async block wait.

"async and await make promises easier to write"

async makes a function return a Promise

await makes a function wait for a Promise

The word “**async**” before a function means one simple thing: a function always returns a promise.

2. The keyword “**await**” makes **JavaScript** wait until that promise settles and returns its result.

Example:

```
async function f() {

let promise = new Promise((resolve, reject) => {
  setTimeout(() => resolve("done!"), 1000)
});

let result = await promise; // wait until the promise resolves (*)

console.log(result); // "done!"
}

f();
```

57 what is AJAX?

Ans:

- Read data from a web server - after a web page has loaded
- Update a web page without reloading the page

- Send data to a web server - in the background

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

- 1. An event occurs in a web page (the page is loaded, a button is clicked)
 - 2. An XMLHttpRequest object is created by JavaScript
 - 3. The XMLHttpRequest object sends a request to a web server
 - 4. The server processes the request
 - 5. The server sends a response back to the web page
 - 6. The response is read by JavaScript
-
- 7. Proper action (like page update) is performed by JavaScript

Example:

```
<!doctype html>

<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Javascript Programs</title>
    <!-- css -->
    <!-- <link rel="stylesheet" href="form.css"> -->
  </head>

  <body>

    <!-- javascript code starts here -->
    <div id="demo">
      <h2>The XMLHttpRequest Object</h2>
      <button type="button" onclick="loadDoc()">Change Content</button>
    </div>

    <script>
```

```

function loadDoc() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("demo").innerHTML =
                this.responseText;
        }
    };
    xhttp.open("GET", "ajax_info.txt", true);
    xhttp.send();
}
</script>

<body>

<!-- javascript code ends here -->

<!-- <script type="text/javascript" src="form.js"></script> -->
</body>

</html>

```

58 what is generators?

Ans: **Generators** are a special class of functions that simplify the task of writing iterators. A **generator** is a function that produces a sequence of results instead of a single value, i.e you generate a series of values.

ES6 introduced a new way of working with functions and iterators in the form of Generators A generator is a function that can stop midway and then continue from where it stopped.

Example:

```

function* g(){ //or function *g(){
    console.log("First");
    yield 1;
    console.log("second");
    yield 2;
    console.log("third");
}
let generator=g();

```

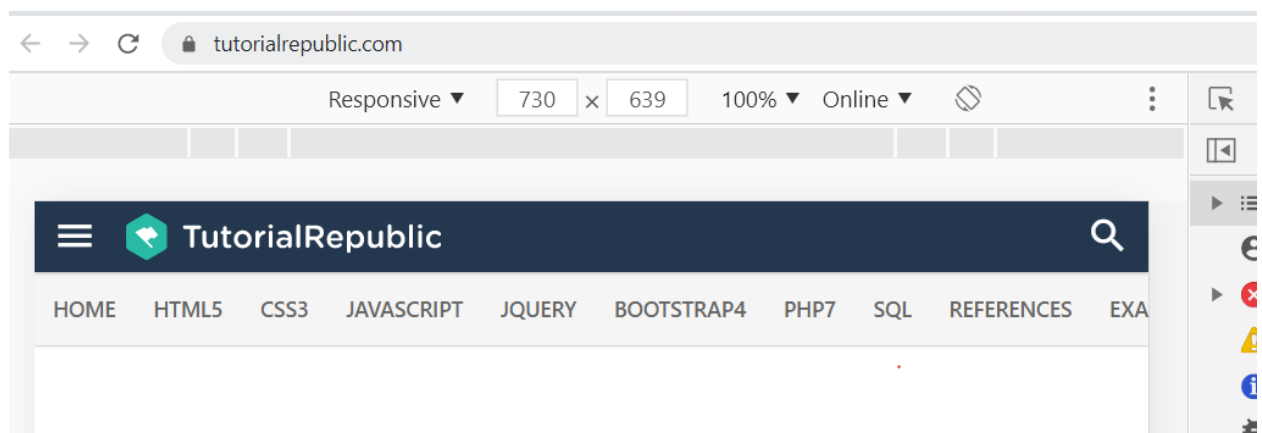


```
generator.next(); //first
generator.next(); //second
```

59 write on say syntax?

When click on button navigate to different url,how will do that in java script?

```
function pageRedirect() {
    window.location.replace("https://www.tutorialrepublic.com/");
}
setTimeout("pageRedirect()", 3000);
```



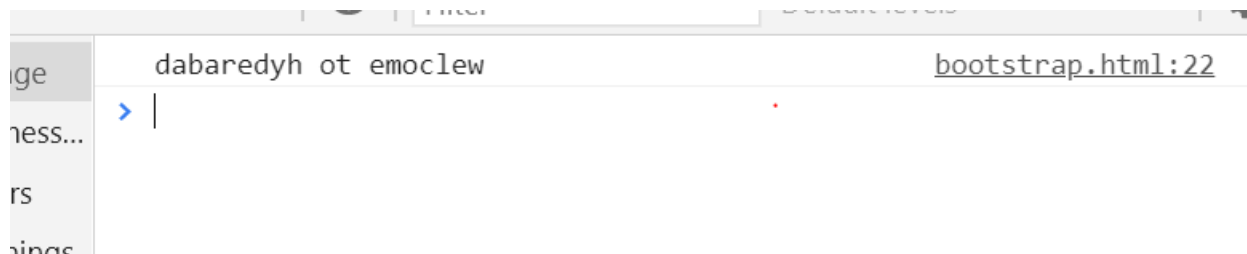
60 write a program to reverse a string?

Ex:str="welcome to hyderabad"

Result: emoclew of dabaredyH

```
var str = "welcome to hyderabad"

console.log(str.split('').reverse().join(''));
```



61 write a program to reverse the sentence?

Ex: "hello world"

Result: dlrow olleh

```
var str = "hello world"

console.log(str.split('').reverse().join(''));
```

The image shows a code editor with the JavaScript code for reversing a string. Below the code, a screenshot of a web browser's developer console is shown. The console displays the output 'dlrow olleh' in a message box, indicating the string 'hello world' has been reversed. The console also shows the file path 'bootstrap.html:22'.

62 write a program for factorial of a number?

```
var num =prompt("enter a number");
fact = 1;
function Fun1() {

for(i=1; i<=num;i++){
    fact *= i; //fact = fact*i 3*2*1=6
}
console.log(fact);
}
Fun1()
```

o/p:

63 write a function to add two numbers?

```
Array = [1,2]
var sum = Array.reduce((a,b)=>{
    return a+b;
})
console.log(sum) //3
```

64 what is call back function ,write syntax?

Ans:

=> CALLBACK-HELL() : It means Executing multiple Asynchronous operations one after another.

This is a big issue caused by coding with complex nested callbacks. Here, each and every callback takes an argument that is a result of the previous callbacks.

some peoples calls it pyramid of dhoom.

we can avoid this, by using promises, async-await

```
const verifyUser = function(username, password, callback){
  DataBase.verifyUser(username, password, (error, userInfo) => {
    if (error) {
      callback(error)
    }else{
      DataBase.getRoles(username, (error, roles) => {
        if (error){
          callback(error)
        }else {
          DataBase.logAccess(username, (error) => {
            if (error){
              callback(error);
            }else{
              callback(null, userInfo, roles);
            }
          })
        }
      })
    }
  })
}
```

65 TEXTBOX 1

TEXTBOX 2

Click

Button

When click on the button the first TEXTBOX value go to second TEXTBOX and second TEXTBOX value Go to first TEXTBOX Write a logic?

66 what is difference between java script and jquery?

Ans: The main **difference is** that **JavaScript is** client-side, i.e., **in the** browser scripting language, whereas **jQuery is** a library (or framework) built with **JavaScript**. ... At the same time, **jQuery is a JavaScript** framework that makes life easier for people who want to program for the browser.

67 how to make a AJAX call by java script/Jquery/angular?

Ans:

In JS:

```
var id = empid;

$.ajax({
  type: "POST",
  url: "../Webservices/EmployeeService.asmx/GetEmployeeOrders",
  data: "{empid: " + empid + "}",
  contentType: "application/json; charset=utf-8",
  dataType: "json",
  success: function(result){
    alert(result.d);
    console.log(result);
  }
});
```

In Jquery:

```
$.ajax({
  url: 'your url',
  type: 'POST', // http method
  data: { myData: 'This is my data.' }, // data to submit
  success: function (data, status, xhr) { // after success your get data
    $('p').append('status: ' + status + ', data: ' + data);
  },
  error: function (jqXHR, textStatus, errorMessage) { // if any error come then
    $('p').append('Error' + errorMessage);
  }
});
```

68 how to fetch data from API in java script?

Example:

```
fetch("https://jsonplaceholder.typicode.com/users")
```

```

.then(x=>{ return x.json();}) //data converted into json format and return
this data
.then(y=>{ console.log(y)}) //catching the return data..
.catch(z=>{console.log("this is error")}) // catch the error..

```

o/p:

```

bootstrap.html:22
▼ (10) [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}] ⓘ
  ▶ 0: {id: 1, name: "Leanne Graham", username: "Bret", email: "...
  ▶ 1: {id: 2, name: "Ervin Howell", username: "Antonette", emai...
  ▶ 2: {id: 3, name: "Clementine Bauch", username: "Samantha", e...
  ▶ 3: {id: 4, name: "Patricia Lebsack", username: "Karianne", e...
  ▶ 4: {id: 5, name: "Chelsey Dietrich", username: "Kamren", ema...
  ▶ 5: {id: 6, name: "Mrs. Dennis Schulist", username: "Leopoldo...
  ▶ 6: {id: 7, name: "Kurtis Weissnat", username: "Elwyn.Skiles"...
  ▶ 7: {id: 8, name: "Nicholas Runolfsson", username: "Maxi...
  ▶ 8: {id: 9, name: "Glenna Reichert", username: "Delphine", em...
  ▶ 9: {id: 10, name: "Clementina DuBuque", username: "Moriah.St...
    length: 10
  ▶ __proto__: Array(0)

```

>

69 what is difference between delete() and splice()?

Ans: **delete** only removes the object from the element **in the** array, the length of the array won't change. **Splice** removes the object and shortens the array.

```

var objects = [10, 54 , 'b' , 3 , 544 , 'fruit' , 8852 , 'Dae' , 2334 , 129 ];
objects.length; // return 10
delete objects[3]; // return true
objects.length; // return 10

console.log(objects)
/* items will be equal to [10, 54 , 'b' , undefined 1 , 544 , 'fruit' , 8852 , 'D
ae' , 2334 , 129 ] */

```

```

var objects2 = [10, 54 , 'b' , 3 , 544 , 'fruit' , 8852 , 'Dae' , 2334 , 129 ];
objects2.length; // return 10
objects2.splice(3,1) ;
objects2.length; // return 9

console.log(objects2)
/* items will be equal to [10, 54 , 'b' , 544 , 'fruit' , 8852 , 'Dae' , 2334 , 129 ]; */

```

```

Array.splice(position,num);

```

Code language: JavaScript (javascript)

The `position` specifies the position of the first item to delete and the `num` argument determines the number of elements to delete.

The `splice()` method changes the original array and returns an array that contains the deleted elements.

70 how can you provide security in your application using java script?

```

71 obj = [
{
  "id":101,
  "name":"amith"
},
{
  "id":102,
  "name":"sumit"
}
.
.
.
.
Multiple objects are there
]

```

I want to display person name whose id is 4000 in console.

Write syntax ?

```
obj = [  
{id: 101,name: "abc"},  
{id: 102,name: "def"},  
{id: 103,name: "ghi"},  
  
]  
  
for(let x of obj){  
  console.log(x.name)  
  console.table(x)  
}
```

72. write a program to palindrome of a number in javascript ?

Ans:

MADAM

RACECAR

121

POP

353

```
// program to check if the string is palindrome or not  
  
function checkPalindrome(str) {  
  
  // find the length of a string  
  const len = string.length;  
  
  // loop through half of the string  
  for (let i = 0; i < len / 2; i++) {  
  
    // check if first and last string are same  
    if (string[i] !== string[len - 1 - i]) {  
      return 'It is not a palindrome';  
    }  
  }  
  return 'It is a palindrome';  
}
```

```
// take input
const string = prompt('Enter a string: ');

// call the function
const value = checkPalindrome(string);

console.log(value);
```

```
// program to check if the string is palindrome or not

function checkPalindrome(str) {

// // convert string to an array
// const arrayValues = string.split('');

// // reverse the array values
// const reverseArrayValues = arrayValues.reverse();

// // convert array to string
// const reverseString = reverseArrayValues.join('');
var reverseString = string.split('').reverse().join('')

if(string == reverseString) {
    console.log('It is a palindrome');
}
else {
    console.log('It is not a palindrome');
}

}

//take input
const string = prompt('Enter a string: ');

checkPalindrome(string);
```

1) Remove duplicates from an array using a Set

A [Set](#) is a collection of unique values. To remove duplicates from an [array](#):

- First, convert an array of duplicates to a `Set`. The new `Set` will implicitly remove duplicate elements.
- Then, convert the `set` back to an array.

```
//remove duplicate elements from array and return new array
let chars = ['A', 'B', 'A', 'C', 'B'];
let uniqueChars = [...new Set(chars)];

console.log(uniqueChars);

var num = [1, 1,2,2, 3, 4,4, 5, 6, 7]

var duplicate = [...new Set(num)]

console.log(duplicate)
```

```

  ▶ (3) ["A", "B", "C"] bootstrap.html:31
  ▶ (7) [1, 2, 3, 4, 5, 6, 7] bootstrap.html:38
> |
```

```
let chars = ['A', 'B', 'A', 'C', 'B'];
debugger;

let uniqueChars = chars.filter((c, index) => {
    return chars.indexOf(c) === index;
});

console.log(uniqueChars);
```

```

  ▶ (3) ["A", "B", "C"] . bootstrap.html:33
  ▶
```

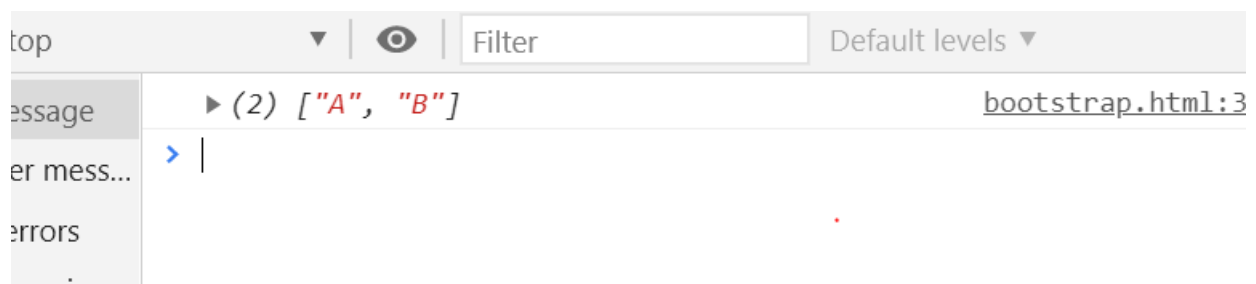
Display the duplicates elements

```
let chars = ['A', 'B', 'A', 'C', 'B'];
debugger;

let uniqueChars = chars.filter((c, index) => {
    return chars.indexOf(c) !== index;
});

console.log(uniqueChars);
```

o/p:



73. arr= [10,20,15,44,99,1,81,101,25,110]

ARRAY REDUCE:

The `reduce()` method reduces the array to a single value.

The `reduce()` method executes a provided function for each value of the array (from left-to-right).

The return value of the function is stored in an accumulator (result/total).

Note: `reduce()` does not execute the function for array elements without values.

Note: This method does not change the original array.

***It has 2 arguments .. one callback, 2nd optional**

***Callback has 2 arguments**

***(x,ele) ele is iteration of an array..**

```
const array = [29.76, 41.85, 46.5];

const sum = array.reduce((total, amount) => total + amount);

console.log(sum) // 118.11
```

```
array = [0,1,2,3,4]
var sum = array.reduce((accumulator, currentValue, currentIndex, array) => {
  return accumulator + currentValue
}, 10)

console.log(sum) // 20
```

<i>callback iteration</i> n	<i>accumul ator</i>	<i>currentValue</i>	<i>currentInd ex</i>	<i>array</i>	return value
first call	10	0	0	[0, 1, 2, 3, 4]	10
second call	10	1	1	[0, 1, 2, 3, 4]	11
third call	11	2	2	[0, 1, 2, 3, 4]	13

fourth call	13	3	3	[0, 1, 2, 3, 4]	16
fifth call	16	4	4	[0, 1, 2, 3, 4]	20

Sum:

```
array = [2,3,5,6]
var sum = array.reduce( (accumulator, currentValue, currentIndex, array) => {
  debugger;

  return accumulator+currentValue;

})

console.log(sum) // 16
```

Largest num:

```
array = [10,11,2,3,444,5,6]
var max = array.reduce( (accumulator, currentValue, currentIndex, array) => {
  debugger;
  if(accumulator>currentValue){
    return accumulator;
  }
  else
  {
    return accumulator = currentValue;
  }

})

console.log(max) // 444
```

Smallest num:

```
array = [10,11,2,3,444,5,6]
var min = array.reduce( (accumulator, currentValue, currentIndex, array) => {
  debugger;
  if(accumulator<currentValue){
    return accumulator;
  }
  else
  {
    return accumulator = currentValue;
  }
})

console.log(min) // 2
```

Find the longest number from the array?

```
arr = [10,20,15,44,99,1,81,101,25,110]

console.log(Math.max(...arr))
console.log(Math.min(...arr))

array = [10,20,15,44,99,1,81,101,25,110]
var largest= 0;

for (i=0; i<=largest;i++){
  if (array[i]>largest) {
    var largest=array[i];
  }
}

console.log(largest);

//largest

var array2 = [3 , 6, 2, 56, 32, 5, 89, 32],
```

```

largest = array2.sort((a,b)=>a-b).reverse()[0];

console.log(largest)

//second largest
array2.sort(function(a, b) {
  return a - b;
});
largest = array[array2.length - 2];
console.log(largest)

```

Elements	Console	Sources	Memory	Application	»	⚙	⋮	➤
top	▼	👁	Filter	Default levels ▼		⚙		
messages	110							bootstrap.html:23
user mess...	1							bootstrap.html:24
to errors	110							bootstrap.html:37
to warnings	89							bootstrap.html:42
info	81							bootstrap.html:49
to verbose	>							

Filter

The `filter()` method **creates a new array** with all elements that pass the test implemented by the provided function.

```

const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter(word => word.length > 6);

console.log(result);
// expected output: Array ["exuberant", "destruction", "present"]

```

Find out Even numbers :

```
var num = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 10, 11, 12, 13, 27]
```

```
even=num.filter(eve=>eve%2==0)
document.write("only even numbers are" +even+ "<br>")
console.log(even)
```

```
▶ (7) [2, 4, 6, 8, 0, 10, 12] bootstrap.html
> |
```

Find out Odd numbers:

```
var num = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 10, 11, 12, 13, 27]
```

```
odd=num.filter(eve=>eve%2 !== 0)
document.write("only odd numbers are" +odd+ "<br>")
console.log(odd)
```

```
▶ (8) [1, 3, 5, 7, 9, 11, 13, 27]
> |
```

Find out the duplicates..

74.how can you copy object in javascript ?

Ans:

To copy an object in JavaScript, you have three options:

1. Use the spread (...) syntax
2. Use the `Object.assign()` method
3. Use the `JSON.stringify()` and `JSON.parse()` methods

Ex:

```
const person = {
```

```

    firstName: 'John',
    lastName: 'Doe'
  });

  // using spread ...
  let p1 = {
    ...person
  };

  console.log(p1);

  // using Object.assign() method
  let p2 = Object.assign({}, person);
  console.log(p2);
  // using JSON
  let p3 = JSON.parse(JSON.stringify(person));
  console.log(p3)

```

o/p:

	Filter	Default levels ▾
es	▶ {firstName: "John", lastName: "Doe"}	bootstrap.html:31
ss...	▶ {firstName: "John", lastName: "Doe"}	bootstrap.html:35
	▶ {firstName: "John", lastName: "Doe"}	bootstrap.html:38
ngs	>	

10 Ways to copy array

```

const sheeps = ['one', 'two', 'three'];

// Old way
const cloneSheeps = sheeps.slice(0,2);

// ES6 way

```



```
const cloneSheepsES6 = [...sheeps];
```

```
console.log(cloneSheeps)
```

```
console.log(cloneSheepsES6)
```

ges	▶ (2) ["one", "two"]	bootstrap.html:28
ess...	▶ (3) ["one", "two", "three"]	bootstrap.html:29
s	>	.

How to clone an array in JavaScript

```
[...]  
for()  
while()  
map()  
filter()  
reduce()  
slice()  
parse |> stringify  
concat()  
from()
```

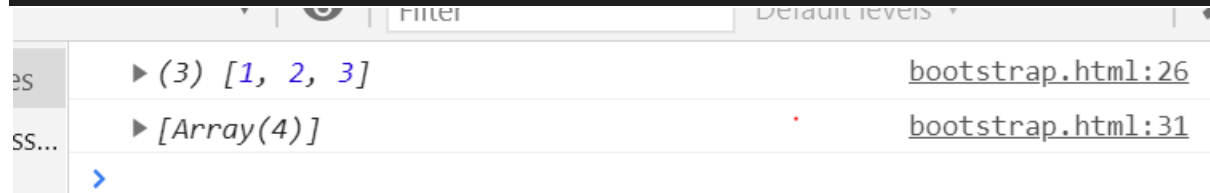
1. Spread Operator (Shallow copy)

```
numbers = [1, 2, 3];  
numbersCopy = [...numbers];  
console.log(numbersCopy)
```

```
numbers = [1, 2, 3];  
numbersCopy = [];  
console.log(numbers)
```

```
numbers.push(4);
```

```
numbersCopy.push(numbers)
console.log( numbersCopy);
```



2. Good Old for() Loop (Shallow copy)

```
numbers = [1, 2, 3];
numbersCopy = [];

for (i = 0; i < numbers.length; i++) {
  numbersCopy[i] = numbers[i];
}

console.log(numbersCopy);
```

3. Good Old while() Loop (Shallow copy)

```
numbers = [1, 2, 3];
numbersCopy = [];
i = -1;

while (++i < numbers.length) {
  numbersCopy[i] = numbers[i];
}

console.log(numbers, numbersCopy)
```

4. Array.map (Shallow copy)

```
numbers = [1, 2, 3];
double = (x) => x * 2;

var a = numbers.map(double);

console.log(numbers)
console.log(a) //2,4,6

single = (y) => y;
```

```
var b = numbers.map(single)
console.log(b) //1,2,3
```

5. Array.filter (Shallow copy)

```
numbers = [1, 2, 3];
numbersCopy = numbers.filter(() => true);

console.log(numbersCopy) //1,2,3

var copy = numbers.filter((y)=> y%2 === 0 )
console.log(copy) //2
```

6. Array.reduce (Shallow copy)

```
numbers = [1, 2, 3];

numbersCopy = numbers.reduce((newArray, element) => {
  newArray.push(element);

  return newArray;
}, []);
console.log(numbersCopy) //1,2,3
```

7. Array.slice (Shallow copy)

```
numbers = [1, 2, 3, 4, 5];
numbersCopy = numbers.slice();
// [1, 2, 3, 4, 5]

console.log(numbersCopy)
```

8. JSON.parse and JSON.stringify (Deep copy)

JSON.stringify turns an object into a string.
JSON.parse turns a string into an object.

```
numbers = [1, 2, 3, 4, 5];
```

```
numbersCopy = JSON.parse(JSON.stringify(numbers));  
  
console.log(numbersCopy); // [1,2,3,4,5]
```

9. Array.concat (Shallow copy)

`concat` combines arrays with values or other arrays.

```
numbers = [1, 2, 3];  
  
var a = numbers.concat(); // [1, 2, 3]  
console.log(a)  
  
var c = numbers.concat(4); // [1, 2, 3, 4]  
var d = numbers.concat(4,5); // [1, 2, 3, 4, 5]  
  
console.log(c,d)
```

10. Array.from (Shallow copy)

This can turn any iterable object into an array. Giving an array returns a shallow copy.

```
numbers = [1, 2, 3];  
numbersCopy = Array.from(numbers);  
  
console.log(numbersCopy) // [1, 2, 3]
```

Mutable vs Immutable Data Types

Mutable:

- object
- array
- function

Immutable:

All primitives are immutable.

- string
- number
- boolean
- null
- undefined
- symbol

75. var x = {name:"amith"}

Var y = x;

Var y = "scott";

Console.log(x.name) ?

Console.log(y.name) ?

```
var x = {name:"amith"};
var y = x;
console.log(y)
var y = "scott";
console.log(x.name);
console.log(y.name);
```

We can update the object with the help of object only not string ...

o/p:

ges	▶ {name: "amith"}	bootstrap.html:22
ness...	amith	bootstrap.html:24
rs	undefined	bootstrap.html:25
things	>	

76. how can you create form in javascript ?

Name :

Age:

City:

Save Edit

Write syntax?

```
<script>
    function validateform(){
        var name=document.myform.name.value;
        var password=document.myform.password.value;

        if (name==null || name==""){
            alert("Name can't be blank");
            return false;
        }else if(password.length<6){
            alert("Password must be at least 6 characters long.");
            return false;
        }
    }
</script>
<body>
    <form name="myform" method="post" action="abc.jsp" onsubmit="return validateform()" >
        Name: <input type="text" name="name"><br/>
        Password: <input type="password" name="password"><br/>
        <input type="submit" value="register">
    </form>
```

77. what are the new features in ES6 ?

ANS:

The let keyword

The const keyword

JavaScript Arrow Functions

JavaScript For/of

JavaScript Classes

JavaScript Promises

JavaScript Symbol

Default Parameters

Function Rest Parameter

`Array.find()`

`Array.findIndex()`

New Number Properties

New Number Methods

New Global Methods

JavaScript Modules

- [arrows](#)
- [classes](#)
- [enhanced object literals](#)
- [template strings](#)
- [destructuring](#)
- [default + rest + spread](#)
- [let + const](#)
- [iterators + for..of](#)
- [generators](#)
- [unicode](#)
- [modules](#)
- [module loaders](#)
- [map + set + weakmap + weakset](#)
- [proxies](#)
- [symbols](#)
- [subclassable built-ins](#)
- [promises](#)
- [math + number + string + array + object APIs](#)
- [binary and octal literals](#)
- [reflect api](#)
- [tail calls](#)

78 Difference between arrow function and normal function ?

Ans:

ES6 version of TypeScript provides an arrow function which is the **shorthand** syntax for defining the anonymous function, i.e., for function expressions.

It omits the function keyword. We can call it fat arrow (because -> is a thin arrow and => is a "fat" arrow). It is also called a **Lambda function**. The arrow function has lexical scoping of "this" keyword.

```
hello = () => {  
  return "Hello World!";  
}  
console.log(hello())
```

Functions

A **JavaScript function** is a block of code designed to perform a particular task.

*Javascript functions are used to perform operations.

*we can call javascript function many times to reuse the code.

A function is the logical blocks of code designed to perform a particular task.. Like JavaScript, TypeScript can also be used to create functions either as a **named function** or as an **anonymous function**.

Functions ensure that our program is readable, maintainable, and reusable. A function declaration has a function's name, return type, and parameters.

```
// ES5: Without arrow function  
var myAdd1 = function(a,b) {  
  return a + b ;  
};  
  
console.log(myAdd1(10,20)) //30  
  
// ES6: With arrow function  
// Anonymous function  
let myAdd = (x, y) => {  
  return x + y;  
}  
  
console.log(myAdd(5,3)); //8
```

79 what is rest and spread operator ?

Ans:

Spread operator: allows iterables(arrays / objects / strings) to be expanded into single arguments/elements..This allows to add an elements into an existing array.

The **spread operator** is a new addition to the set of **operators** in JavaScript ES6. It takes in an iterable (e.g an array) and expands it into individual elements. The **spread operator** is commonly used to make shallow copies of JS objects.

Spread operator Example:

```
let arr1 = ['A', 'B', 'C'];  
  
let arr2 = ['X', 'Y', 'Z'];  
  
let result = [...arr1, ...arr2];  
  
console.log(result); // ['A', 'B', 'C', 'X', 'Y', 'Z']
```

```
var number1 = [1,2,3];  
var number2 = [4,5,6];  
var merged = [...number1, ...number2];  
console.log(merged)
```



The spread operator is used to initialize arrays and objects from another array or object. We can also use it for object de-structuring. It is a part of the ES 6 version.

Rest parameter: collects all remaining elements into an array.

Collects all the remaining elements in to an array. (...) is the identifier of rest parameter.:

gathers the rest of the list of arguments into an array. When **three dots (...)**

```
function fun(...input){  
  let sum = 0;  
  for(let i of input){  
    sum+=i;  
  }  
  return sum;  
}  
  
console.log(fun(1,2)); //3  
console.log(fun(1,2,3)); //6  
console.log(fun(1,2,3,4,5)); //15
```

```
console.log("Rest Operator")
```

```
var sno = [10, 20, 30, 40, 50]
var [a, b, ...A] = sno

console.log("a is ",a);
console.log(b);
console.log(A[0]);
console.log(A[1]);
console.log(A)
```

80 what is callbackhell ?

Ans:

=> **CALLBACK-HELL()** : It means Executing multiple Asynchronous operations one after another.

This is a big issue caused by coding with complex nested callbacks. Here, each and every callback takes an argument that is a result of the previous callbacks.

some peoples calls it pyramid of dhoom.

we can avoid this, by using promises, async-await

81.what is set timeout?

Ans: **setTimeout** is a method of the global window object. It executes the given function (or evaluates the given string) after the time given as second parameter passed.

The **setTimeout()** method calls a function or evaluates an expression after a specified number of milliseconds.

=> **setTimeout()** : this method will execute the given function, after waiting a specified number of milliseconds.

syntax : `window.setTimeout(function, milliseconds);`

=> **setIntervale()** : this method will repeat the given function at every given time interval.

syntax : `window.setInterval(function, milliseconds)`

Example:

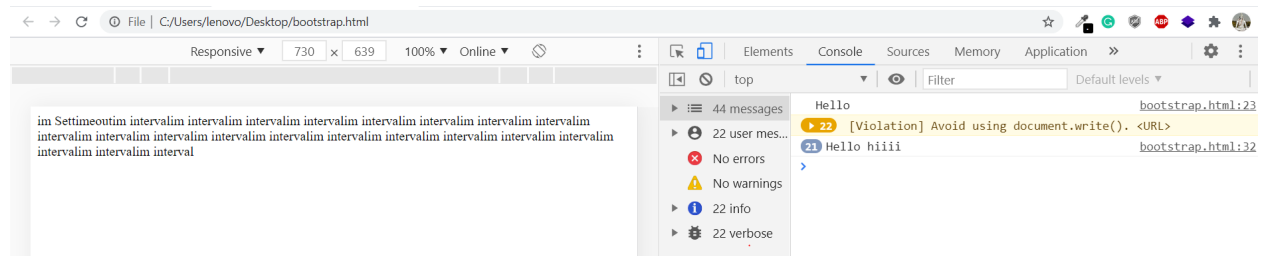
```
function myFunction() {
  setTimeout(function(){
    console.log("Hello");
    document.write("im Settimeout")
  }, 3000);
}
myFunction();

function myFunction1() {
  setInterval(function(){
    console.log("Hello hiii");
    document.write("im interval")
  }, 3000);
}
myFunction1();
```

o/p:

after 3 sec

Hello



82. what is difference between callback, callback hell and promise ?

Ans:

=> CALLBACK() : It is a function, executing itself when associated task execution has been completed successfully. Passing function as argument to another function.

we can pass two arguments through callback those are error and result.

A **callback** is a function passed as an argument to another function.

```
// function
function greet(name, callback) {
  console.log('Hi' + ' ' + name);
```

```

    callback();
}

// callback function
function callMe() {
    console.log('I am callback function');
}

// passing function as an argument
greet('Peter', callMe);

```

o/p:



Ex:

setTimeout()

=> CALLBACK-HELL() : It means Executing multiple Asynchronous operations one after another.

This is a big issue caused by coding with complex nested callbacks. Here, each and every callback takes an argument that is a result of the previous callbacks.

some peoples calls it pyramid of dhoom.

we can avoid this, by using promises, async-await

=> PROMISES : promises are worked with asynchronous operations and it returns only one value either promise-resolve or promise-reject.

```

//we used x,y instead of resolve and reject
var p1 = new Promise(function(resolve,reject){

```

```

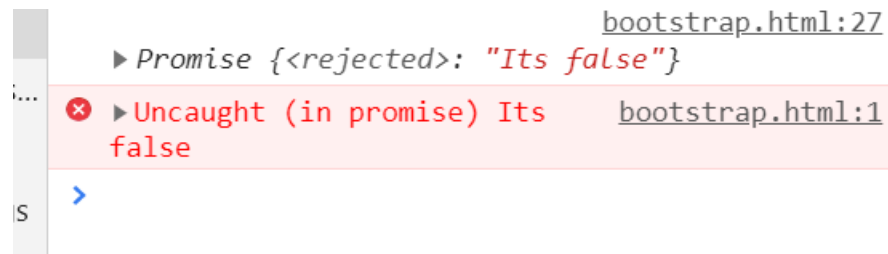
        var res = false;
        if(res){ resolve("its true");}
        else{ reject("Its false");}
    })

console.log(p1);

//stored promise into p1
p1.then(function(val){console.log(val + "----->it is res resolve value");})
.catch(function(val) {console.log(val);})

```

o/p:



The screenshot shows a web browser's developer console. At the top, a message indicates a Promise rejection: `bootstrap.html:27` `Promise {<rejected>: "Its false"}`. Below this, a red error message is displayed: `Uncaught (in promise) Its false` at `bootstrap.html:1`. A blue arrow points to the error message.

```

var promise = new Promise(function(resolve, reject) {
    const x = "geeksforgeeks";

    const y = "geeksforgeeks"
    if(x === y) {
        resolve();
        console.log(x)
    } else {
        reject();
    }
});

```

=> **OBSERVABLES** : It means A sequence of data that are arriving asynchronously from the server over a time with help of subscribe method.

here we use `http-cal` and `http-response`.

import {Observable} from 'rxjs' package.

synchronous code is executed in sequence – each statement waits for the previous statement to finish before executing.

Asynchronous code doesn't have to wait – your program can continue to run.

It won't carry previous req and res data with current request..

synchronous calls:

Synchronous JavaScript: As the name suggests synchronous means to be in a sequence, i.e. every statement of the code gets executed one by one. So, basically a statement has to wait for the earlier statement to get executed.

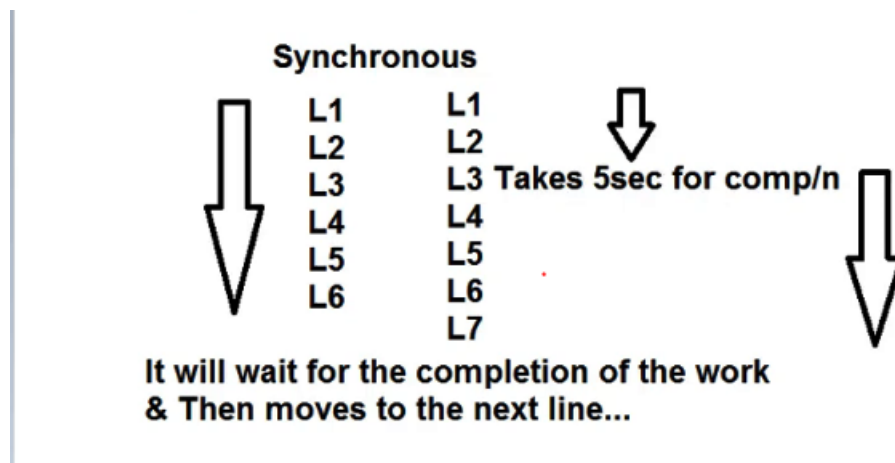
11

12-- > 10sec

13

14

.
. .
.



Asynchronous calls:

Asynchronous JavaScript: Asynchronous code allows the program to be executed immediately where the synchronous code will block further execution of the remaining code until it finishes the current one.

11 --> Asynchronous call – 10sec

Result: true work1 false work2

12

13

14

Asynchronous

L1
L2
L3 Takes 5sec.
L4
L5
L6
L7



It doesn't wait for completion of a work.
Moves to the next line for compiling.

```
console.log("***** synchronous")
// synchronous starts////////////////////////////////////

//instead of api ..we create function
function GetUserInfo(){
    // huge set of records(5 lacks) - takes time
    const dt = Date.now() + 3000;
    console.log("after 5 sec")
    while(Date.now() < dt){
        continue;
    }
    console.log("Work2");
}

console.log("Work1");
//Calls API for data - takes 5 sec
```



```

GetUsersInfo();
  console.log("Work3");

  console.log("***** Asynchronous")

//   synchronous ends////////////////////////////////////

//   Asynchronous starts////////////////////////////////////
console.log("Async_Work1");
  //Calls API for data - takes 5 sec
  //callback function
  setTimeout(function(){
    console.log("Async_Work2");

    },5000)

  console.log("Async_Work3");
  console.log("after 5 seconds..")

//   Asynchronous ends////////////////////////////////////

```

o/p:

S	*****	bootstrap.html:20
	synchronous	
S...	Work1	bootstrap.html:35
	after 5 sec	bootstrap.html:27
S	Work2	bootstrap.html:32
	Work3	bootstrap.html:39
	*****	bootstrap.html:41
	Asynchronous	
	Async_Work1	bootstrap.html:46
	Async_Work3	bootstrap.html:54
	after 5 seconds..	bootstrap.html:55
	Async_Work2	bootstrap.html:50
	>	.

83.what are for.. of and for ..in loop?

Ans:

For in – it's an index based iteration

-works for iterables / complex objects

For of- doesn't provide the index..only provide the values..

-works for iterables only

Difference is :

It takes const. by using const variable . for of..very powerfull

that is main power of..for of..

1. Working with arrays

```
//1.for in
var arr = [1,2,3,4,5];
console.log("for in loop:")
for(var x in arr){
  //console.log(x);
  console.log(x);
}

// 2 for of

var arr = [1,2,3,4,5];
console.log("for of loop:")
for(var x of arr){
  console.log(x);
}
```

o/p:

for in loop:
0
1
2
3
4
for of loop:
1
2
3
4
5
>

Use arr[x] for in loop

```
//1.for in
var arr = [1,2,3,4,5];
console.log("for in loop:")
for(var x in arr){
//console.log(x);
console.log(arr[x]);
}

// 2 for of

var arr = [1,2,3,4,5];
console.log("for of loop:")
for(var x of arr){
console.log(x);
}
```

sages	for in loop:
mes...	1
ors	2
nings	3
	4
	5
oose	for of loop:
	1
	2
	3
	4
	5
	>

```
// // 3. objects
// // for in
var obj1 = {"name": "abc", "sal": 50000};
console.log("for in loop with object")
for(var x in obj1){
  console.log(x);
  console.log(obj1[x]);
}

var obj2 = {"name": "abc", "sal": 50000};
console.log("for of loop:")
for(var y of obj2){
  console.log(y);
}
```

ges	for in loop with object	bootstrap.html:27
ess...	name	bootstrap.html:29
	abc	bootstrap.html:30
	sal	bootstrap.html:29
ings	50000	bootstrap.html:30
	for of loop:	bootstrap.html:34
ose	<div> ✖ <div> <div>► Uncaught TypeError: obj2 is not iterable</div> <div>at bootstrap.html:35</div> </div> </div>	
	>	

By using constant variable:

```
//for of
var myarr = [1,2,3,4,5];

console.log("for of loop: with const")
for(const c of myarr){
  console.log(c);
}

//for in
var myarr = [1,2,3,4,5];

console.log("for in loop: with const")
for(const b in myarr){
  console.log(b)
}

//for loop
var myarr = [1,2,3,4,5];

console.log("for loop: with const")
for(const x = 0; x<myarr.length; x++){
  console.log(x);
}
```

top	Filter	Default levels
5 messages	for of loop: with const	bootstrap.html:25
4 user mes...	1	bootstrap.html:27
error	2	bootstrap.html:27
No warnings	3	bootstrap.html:27
4 info	4	bootstrap.html:27
No verbose	5	bootstrap.html:27
	for in loop: with const	bootstrap.html:34
	0	bootstrap.html:36
	1	bootstrap.html:36
	2	bootstrap.html:36
	3	bootstrap.html:36
	4	bootstrap.html:36
	for loop: with const	bootstrap.html:44
	0	bootstrap.html:46
	✖ ▶ Uncaught TypeError: Assignment to constant variable. at bootstrap.html:45	bootstrap.html:45
	>	

For of :

```
const array = ['hello', 'world', 'of', 'Corona'];

for (const item of array) {
  console.log(item);
}
```

For in :

```
const object = {a: 1, b: 2, c: 3};

for (const property in object) {
  console.log(`${property}: ${object[property]}`);
}
```

84. what is rest API ?

Ans:

An **API** acts as an interface between two different applications so that they can communicate with each other.

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you're using an **API**.

API : api service is type of distributed method use to share data of applications business logics with hydrogenous platforms

Api services are shareable we can access those service through URL address of service.

Ex:

..

A REST API “representational state transfer” (also known as RESTful API) is an application programming interface (API or web API) .. A REST API is a standardized architecture style for creating a Web Service API.

An API is a set of definitions and protocols for building and integrating application software.

Or

it's an architectural style of developing applications where client and server act as separate entities..

maintaining complete statelessness..

Applications following rest architectural style are known as Restful applications

Webservice:

Introduction to RESTful web services

A web service is a collection of open protocols and standards used for exchanging data between applications or systems.

Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

Web services based on REST Architecture are known as RESTful web services.

These webservices uses HTTP methods to implement the concept of REST architecture.

A RESTful web service usually defines a URI, Uniform Resource Identifier a service, provides resource representation such as JSON and set of HTTP Methods.

85. How can we handles error in javascript ?

Ans:

Error: A mistake might lead to an error causing the program to produce unexpected results

There are three types of errors in programming: (a) Syntax Errors, (b) Runtime Errors, and (c) Logical Errors.

(a) Syntax Errors,

Syntax errors, also called **parsing errors**, occur at compile time in traditional programming languages and at interpret time in JavaScript.(Ex: keywords,semicolons missing etc..)

What is Exception Handling

In programming, exception handling is a process or method used for handling the abnormal statements in the code and executing them.

For example, the Division of a non-zero value with zero will result into infinity always, and it is an exception. Thus, with the help of exception handling, it can be executed and handled.

In exception handling:

A throw statement is used to raise an exception. It means when an abnormal condition occurs, an exception is thrown using throw.

The thrown exception is handled by wrapping the code into the try...catch block. If an error is present, the catch block will execute, else only the try block statements will get executed.

Thus, in a programming language, there can be different types of errors which may disturb the proper execution of the program.

Types of Errors

While coding, there can be three types of errors in the code:

1. **Syntax Error:** When a user makes a mistake in the pre-defined syntax of a programming language, a syntax error may appear.
2. **Runtime Error:** When an error occurs during the execution of the program, such an error is known as Runtime error.
3. **Logical Error:** An error which occurs when there is any logical mistake in the program that may not produce the desired output, and may terminate abnormally. Such an error is known as Logical error.

Exception Handling Statements

There are following statements that handle if any exception occurs:

- o throw statements
- o try...catch statements
- o try...catch...finally statements.

These exception

Ex:

```
function fn(){
    try{
        return 10;
    }
    catch(e){
        document.write(e);
    }
    finally{
        console.log('emp info');
        return 20;
    }
}

console.log(fn());
```

```
<script type = "text/javascript">
    <!--
        try {
            // Code to run
            [break;]
        }

        catch ( e ) {
            // Code to run if an exception occurs
            [break;]
        }
    </script>
```

```
    [ finally {  
        // Code that is always executed regardless of  
        // an exception occurring  
    }]  
    //-->  
</script>
```

try...catch...finally statements

Finally is an optional block of statements which is executed after the execution of try and catch statements. Finally block does not hold for the exception to be thrown. Any exception is thrown or not, finally block code, if present, will definitely execute. It does not care for the output too.

86. write the logic for the given programme ?

5 4 3 2 1

4 3 2 1

3 2 1

2 1

1

```
for(var i=5; i>=1; i--){  
  
    for(var j=5-i; j>=1; j--){  
  
        document.write("&nbsp;");  
  
    }  
    for(var k=i; k>=1; k--){  
  
        document.write("&nbsp;",k)  
  
    }  
    document.write("<br>")  
}
```

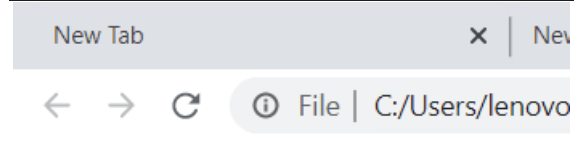
Normal program is :

```
for(var i=1; i<=5; i++){
```

```

for(var j=1; j<=i; j++){
    //debugger;
    document.write("*");
}
document.write("<br>")
}

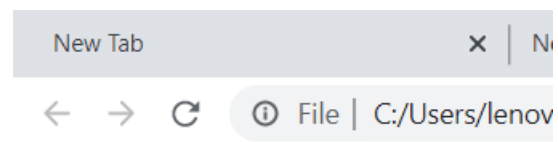
```



```

for(var i=1;i<=5;i++){
    for(var j=5;j>=i;j--){
        document.write("*")
    }
    document.write("<br>")
}

```



```

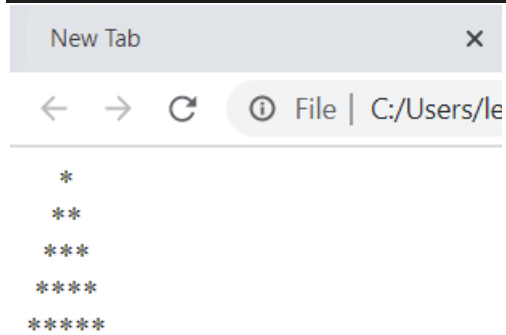
for(var i=1; i<=5; i++){
    for(var j=5; j>i; j--){
        document.write("&nbsp;")
    }
}

```

```

}
for(var k=1; k<=i; k++ ){
    document.write("*")
}
document.write("<br>")
}

```



87 what is default parameter and optional parameter ?

Function **parameters** are the names listed in the function's definition. Function **arguments** are the real values passed to the function. **Parameters** are initialized to the values of the **arguments** supplied.

```

function argumentVar(parameter1, parameter2, parameter3){
    console.log(arguments.length); // Logs the number of arguments passed.
    console.log(arguments[3]); // Logs the 4th argument. Follows array indexing notations.
}

argumentVar(1,2,3,4,5);
// Log would be as follows
// 5
// 4

// 5 is the number of arguments
// 4 is the 4th argument

```

Ans:

Javascript provides an option to set default values to the function parameters.

If the user does not pass a value to an argument, TypeScript initializes the default value for the parameter.

The behavior of the default parameter is the same as an optional parameter.

```
function displayName(name, greeting = "Hello") {  
    return greeting + ' ' + name + '!';  
}  
console.log(displayName('JavaTpoint')); //Returns "Hello JavaTpoint!"  
console.log(displayName('JavaTpoint', 'Hi')); //Returns "Hi JavaTpoint!".  
console.log(displayName('Sachin')); //Returns "Hello Sachin!"
```

```
function multiply(a, b = 1) {  
    return a * b;  
}  
  
console.log(multiply(5, 2));  
// expected output: 10  
  
console.log(multiply(5));  
// expected output: 5
```

o/p:

Hello JavaTpoint!	bootstrap.html:32
Hi JavaTpoint!	bootstrap.html:33
Hello Sachin!	bootstrap.html:34
>	

- **Optional parameter** takes makes value of **parameters** to 'undefined', while **Default parameters** provides **default** value to **parameter** if you don't provide it any value. ... Passing 'null' will cause the **function** to take 'null' as a value of that **parameter**.

a function without passing any arguments in JavaScript those parameters are optional. If we declare the function without passing arguments, then each parameter value is undefined.

```
function showDetails(id,name,e_mail_id) {
  console.log("ID:", id, " Name:",name);
  if(e_mail_id!==undefined)
    console.log("Email-Id:",e_mail_id);
}
showDetails(101,"Virat Kohli");
showDetails(105,"Sachin","sachin@javatpoint.com");
```

o/p:

ID: 101	Name: Virat Kohli	bootstrap.html:21
ID: 105	Name: Sachin	bootstrap.html:21
Email-Id: sachin@javatpoint.com		bootstrap.html:23

Rest parameter

It is used to pass **zero or more** values to a function. We can declare it by prefixing the **three "dot"** characters ('...') before the parameter.

```
function sum(a, ...b) {
  for (var i = 0; i < b.length; i++) {
    a += b[i];
  }
  return a;
}
let result1 = sum(3, 5);
let result2 = sum(3, 5, 7, 9);
console.log(result1 + "\n" + result2);
```

```
function restParam(...restArgs){
  console.log(restArgs.length); // Logs the number of arguments passed
  console.log(restArgs[3]); // Logs the 4th argument
}

restParam(1,2,3,4,5);
// Log would be as follows
// 5
```

```
// 4

// 5 is the number of arguments
// 4 is the 4th argument
```

Rules to follow in rest parameter:

- o We can use only one rest parameter in a function.
- o It must be an array type.
- o It must be the last parameter in a parameter list.

88 what is difference between “null” and “undefined” ?

In JavaScript, undefined is a type, whereas null an object.

Undefined

We can't initialize any value to that variable that is called Undefined except Undefined value

It means a variable declared, but no value has been assigned a value.

For example,

```
var demo;
console.log(demo); //shows undefined
console.log(typeof demo); //shows undefined
```

null

Whereas, null in JavaScript is an assignment value. You can assign it to a variable.

The Null accepts the only one value, which is null.

We can't assign any value to that variable that is called null except null value

For example,

```
var demo = null;
console.log(demo); //shows null
console.log(typeof demo); //shows object
```

89 20 + "30" + 10

o/p:203010

90 "20" + 30 + 10

o/p: 203010

Ans: First of all Javascript compile the programs from left to right. Therefore, "20" is a String and it is in left side. when it is add with 30. It convert "30" to a String. "20" + "30" + 10 => "2030" + 10. and at last it is add with 10. It convert "10" to a String. "20" + "30" + "10" => "203010".

```
var x = "Volvo" + 16 + 4;  
//volvo164 javascript evalute left to right after string no calulations  
console.log(x); //Volovo164  
console.log("20" + 30 + 10); //203010  
console.log(+ "20" + 30 + 10); //60  
console.log( 30 + 10 + "20"); //4020  
console.log( 30 + "20" + 10); //302010
```

91 Is 'false' is false ?

```
console.log('false');//false
```

92 Is '' is false ?

```
console.log(' '); //empty
```

93 Is "" is false ?

```
console.log(""); //empty
```

94 true %1 ?

```
console.log(true%1); //0
```

95 false % 1 ?

```
console.log(false%1); //0
```


96 null == undefined ?

```
console.log(null===undefined); //false  
console.log(null==undefined); //true
```

97 - '34' + 10 ?

```
console.log(-'34'+ 10); // -24  
console.log('-34'+ 10); // -3410
```

98 1 + "2" + "2" ?

```
console.log(1 + "2" + "2" ); // 122
```

99 1 + "1" + "2" ?

```
console.log(1 + "1" + "2" ); // 112
```

100 false == '0' ?

```
console.log(false == '0'); // true ..0==0
```

101 false === '0' ?

```
console.log(false === '0'); // false ..boolean=== string
```

102 1<2<3 ?

Ans: The output is `true` and `false`. Here is the explanation,

In Java script relational operators are evaluated from left to right, **false equals 0**, and **true equals 1** for number comparisons.

So comparison evaluation is something like this,

```
1 < 2 < 3 => true < 3 => 1 < 3 => true
```

```
3 > 2 > 1 => true > 1 => 1 > 1 => false
```

103 3>2>1 ?

Ans: The output is `true` and `false`. Here is the explanation,

In java script relational operators are evaluated from left to right, **false equals 0**, and **true equals 1** for number comparisons.

So comparison evaluation is something like this,

```
3 > 2 > 1 =>  
true > 1 =>  
1 > 1 => false
```

104 type of null ?

```
console.log(typeof(null)); //object
```

105 type of undefined ?

```
console.log(typeof(undefined)); //undefined
```

106 type of {}?

```
console.log(typeof{}); // object
```

107 type of ()?

108 type of [] ?

```
console.log(typeof[]); // object
```

109 type of date ?

```
console.log(typeof(new Date())); //object  
console.log(typeof(Date())); //string
```

110 type of array ?

```
console.log(typeof(Array)); //function
```

111 type of NAN ?

```
console.log(typeof(NaN)); //number
```

Type of the operand	Result
Object	"object"
boolean	"boolean"
function	"function"
number	"number"
string	"string"
undefined	"undefined"

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Javascript Programs</title>
    <!-- css -->
    <!-- <link rel="stylesheet" href="form.css"> -->
  </head>

  <body>
```

```

    <!-- javascript code starts here -->
    <script>

console.log(typeof(NaN)); //function

console.log(typeof(Array)); //function

console.log(typeof(new Date())); //object
console.log(typeof(Date())); //string

console.log(typeof[]); // object

console.log(); //

console.log(typeof{}); // object

console.log(typeof(undefined)); //undefined
console.log(typeof(null)); //object

console.log(typeof(4+7) + "38qsn") //returns number
console.log(typeof("4"+"7" + "39qsn")) //returns string
console.log(typeof(4*"7" + "40 qsn")); //returns number
console.log(typeof(4+"7")); //returns string

// Numbers
typeof 37 === 'number';
typeof 3.14 === 'number';
typeof(42) === 'number';
typeof Math.LN2 === 'number';
typeof Infinity === 'number';
typeof NaN === 'number'; // Despite being "Not-A-Number"
typeof Number('1') === 'number'; // Number tries to parse things into numbers
typeof Number('shoe') === 'number'; // including values that cannot be type coerced to a number

typeof 42n === 'bigint';

// Strings
typeof '' === 'string';
typeof 'bla' === 'string';
typeof `template literal` === 'string';
typeof '1' === 'string'; // note that a number within a string is still type of string

```

```
typeof (typeof 1) === 'string'; // typeof always returns a string
typeof String(1) === 'string'; // String converts anything into a string, safer than toString

// Booleans
typeof true === 'boolean';
typeof false === 'boolean';
typeof Boolean(1) === 'boolean'; // Boolean() will convert values based on if they're truthy or falsy
typeof !(1) === 'boolean'; // two calls of the ! (logical NOT) operator are equivalent to Boolean()

// Symbols
typeof Symbol() === 'symbol'
typeof Symbol('foo') === 'symbol'
typeof Symbol.iterator === 'symbol'

// Undefined
typeof undefined === 'undefined';
typeof declaredButUndefinedVariable === 'undefined';
typeof undeclaredVariable === 'undefined';

// Objects
typeof {a: 1} === 'object';

// use Array.isArray or Object.prototype.toString.call
// to differentiate regular objects from arrays
typeof [1, 2, 4] === 'object';

typeof new Date() === 'object';
typeof /regex/ === 'object'; // See Regular expressions section for historical results

// The following are confusing, dangerous, and wasteful. Avoid them.
typeof new Boolean(true) === 'object';
typeof new Number(1) === 'object';
typeof new String('abc') === 'object';

// Functions
typeof function() {} === 'function';
typeof class C {} === 'function';
typeof Math.sin === 'function';
```

```
</script>
  <!-- javascript code ends here -->

  <!-- <script type="text/javascript" src="form.js"></script> -->
</body>

</html>
```

112 Form1 ,form2,form3

Click on form1 open and fill details then next form will open write logic ?

113 what is return in java script function ?

Ans:

The return statement stops the execution of a function and returns a value from that function.

Access the var a

```
var a = 10;
console.log(a)
```

```
var a = 10;

function fun1(){

  return a
}

console.log(fun1())
```

o/p:

10

with function

```
function fun1(a){  
  
    return a;  
  
}  
  
console.log(fun1(10))
```

```
function fun1(a){  
  
    return a;  
  
}  
  
var x = fun1(10)  
console.log(x)
```

message	10	clouser.html:28
server mess...	>	
errors		
warnings		

Ans: The return **statement** ends function execution and specifies a value to be returned to the function caller.

Ex:

```
function ReactArea(width, height){  
if(width > 0 && height > 0) {  
  
return width * height;  
}  
return 0;  
}  
var x = ReactArea(10,20);  
console.log(x);
```



114 what is destructuring in javascript ES6 ?

Ans:

What is Destructuring?

Destructuring simply implies breaking down a complex structure into simpler parts. In JavaScript, this complex structure is usually an object or an array. With the destructuring syntax, you can extract smaller fragments from arrays and objects.

Object example:

```
const student1 = {
  firstname: 'hyd',
  lastname: 'ts',
  country: 'india'
};

// Object Destructuring
const { firstname, lastname, country } = student1;

console.log(firstname, lastname, country); // Glad Chinda Nigeria

console.log(student1);
```


hyd ts india	clouser.html:30
▶ {firstname: "hyd", lastname: "ts", country: "ind ia"}	clouser.html:32
>	

Array example:

```
<script>

[a,b,c,d,e] = [10,20,30,40,50]

console.log(a,b);

[a,b, ...rest] = [10,20,30,40,50,60]

console.log(rest)

</script>
```

o/p:

10 20	clouser.html:24
▶ (4) [30, 40, 50, 60]	clouser.html:28
>	

115 clouser question

Closure means that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.

Access the global and local variables

```
var a = 10;
function HeeloGreet(){ //outer function
```

```

var b=200;

console.log(a);

console.log(b);
}

HeeloGreet()

```

Clouser:

```

<script>

function HeeloGreet(){ //outer function

    var outer = "angular is outer funtion varaible";
    function Inner(){ // inner funtion
        var inner = "node";

        console.log(outer)
        console.log(inner)
    }
    return Inner()
}

HeeloGreet()

</script>

```

Access the outer function parameters

```

<script>

function HeeloGreet(a,b){ //outer function

```

```

var outer = "angular is outer funtion varaible";
function Inner(){ // inner funtion
    var inner = "node";

    console.log(outer)
    console.log(a+b)
    console.log(inner)
}
return Inner()
}

HeeloGreet(10,20)

</script>

```

o/p:

es	angular is outer funtion varaible	clouser.html:29
ss...	30	clouser.html:30
	node	clouser.html:31
ngs	>	.