JavaScript

**--------------------------------------------Variable--------------------------------------------------**

A variable is a container for a value, like a number we might use in a sum, or a string that we might use as part of a sentence.

**4 Ways to Declare a JavaScript Variable:**

* var
* let
* const
* nothing

**Var –**

var keyword in JavaScript: The var is the oldest keyword to declare a variable in JavaScript.

**Scope -** Global scoped or function scoped. The scope of the var keyword is the global or function scope. It means variables defined outside the function can be accessed globally, and variables defined inside a particular function can be accessed within the function.

|  |
| --- |
| **Example** |
| Variable ‘a’ is declared globally. So, the scope of the variable ‘a’ is global, and it can be accessible everywhere in the program. The output shown is in the console. |
| <script>  var a = 10 // Global scope  function f(){  console.log(a) //function scope  }  f(); // output 10  console.log(a); // output 10  </script> |

**Let –**

let keyword in JavaScript: The let keyword is an improved version of the var keyword.

**Scope:** block scoped: The scope of a let variable is only block scoped. It can’t be accessible outside the particular block **({block}).** Let’s see the below example.

|  |
| --- |
| **Example** |
| <script>  let a = 10;  function f() {  let b = 9  console.log(b);  console.log(a);  }  f(); //output – 9 and 10  </script> |

**Const –**

const keyword in JavaScript: The const keyword has all the properties that are the same as the let keyword, except the user cannot update it.

**Scope:** block scoped: When users declare a const variable, they need to initialize it, otherwise, it returns an error. The user cannot update the const variable once it is declared.

|  |
| --- |
| **Example** |
| We are changing the value of the const variable so that it returns an error. The output is shown in the console. |
| <script>  const a = 10;  function f() {  a = 9  console.log(a)  }  f();  //output  a=9  TypeError:Assignment to constant variable.  </script> |

**Differences between var, let, and const**

|  |  |  |
| --- | --- | --- |
| **var** | **let** | **const** |
| The scope of a var variable is functional scope. | The scope of a let variable is block scope. | The scope of a const variable is block scope. |
| It can be updated and re-declared into the scope. | It can be updated but cannot be re-declared into the scope. | It cannot be updated or re-declared into the scope. |
| It can be declared without initialization. | It can be declared without initialization. | It cannot be declared without initialization. |
| It can be accessed without initialization as its default value is “undefined”. | It cannot be accessed without initialization, as it returns an error. | It cannot be accessed without initialization, as it cannot be declared without initialization. |

**---------------------------------------------------Array--------------------------------------------------**

In JavaScript, array is a single variable that is used to store different elements. It is often used when we want to store list of elements and access them by a single variable. Unlike most languages where array is a reference to the multiple variable.  
**You should use arrays when you want the element names to be numbers.**

**Declaration –**

Two ways to declare an array.  
Generally method 1 is preferred over the method 2. Let us understand the reason for this.

|  |
| --- |
| var House = [ ]; // method 1  var House = new Array(); // method 2 |

But generally method 1 is preferred over the method 2. Let us understand the reason for this.

|  |
| --- |
| **Example** |
| // Method 1  let fruits = ["Apple", "Orange", "Plum"];  alert( fruits[0] ); // Apple  alert( fruits[1] ); // Orange  alert( fruits[2] ); // Plum  // Method 2 - var house = new Array(10, 20, 30, 40, 50); |

**------------------------------------Array Methods-----------------------------------**

**Self Changing:**

1. **Sort() Method**

Sorting an array means to arrange the elements in the array in a certain order.

The arr.sort() method is used to sort the array in place in a given order according to the compare() function. If the method is omitted then the array is sorted in ascending order. Syntax:

**Example –**

|  |
| --- |
| var arr = [2, 5, 8, 1, 4]  document.write(arr.sort());  document.write(arr);  //output  1,2,4,5,8  1,2,4,5,8  **Ascending order array:**  arr.sort((a,b) => {  return a>b ? 1 : -1;  }) console.log(arr);  **Descending order array:**  arr.sort((a,b) => {  return a>b ? -1 : 1;  }) console.log(arr); |

1. **Push() Method**

The arr.push() method is used to push one or more values into the array. This method changes the length of the array by the number of elements added to the array.

* Adds new items to the end of an array.
* Changes the length of the array.
* Returns the new length.

|  |
| --- |
| **Example** |
| var arr = [34, 234, 567, 4];  arr.push(23,45,56);  console.log(arr);  //output  7  34,234,567,4,23,45,56 |

1. **Pop() Method**

The arr.pop() method is used to remove the last element of the array and also returns the removed element. This function decreases the length of the array by 1.

* The pop() method removes (pops) the last element of an array.
* The pop() method changes the original array.
* The pop() method returns the removed element.

|  |
| --- |
| **Example** |
| <script>  const fruits = ["Banana", "Orange", "Apple", "Mango"];  let removed = fruits.pop();  document.getElementById("demo").innerHTML = removed;  </script>  //output  Mango |

1. **Shift() Method**

The arr.shift() method removes the first element of the array thus reducing the size of the original array by 1.

* The shift() method removes the first item of an array.
* The shift() method changes the original array.
* The shift() method returns the shifted element.

|  |
| --- |
| **Example** |
| <script>  const fruits = ["Banana", "Orange", "Apple", "Mango"];  fruits.shift();  document.getElementById("demo").innerHTML = fruits;  </script>  //output  Orange,Apple,Mango |

1. **Unshift() Method**

The arr.unshift() method is used to add one or more elements to the beginning of the given array. This function increases the length of the existing array by the number of elements added to the array.

* The unshift() method adds new elements to the beginning of an array.
* The unshift() method overwrites the original array.

|  |
| --- |
| **Example** |
| <script>  const fruits = ["Banana", "Orange", "Apple", "Mango"];  fruits.unshift("Lemon", "Pineapple");  document.getElementById("demo").innerHTML = fruits;  </script>  //Output  Lemon,Pineapple,Banana,Orange,Apple,Mango |

**New Array:**

1. **Map Method()**

The map() method in JavaScript creates an array by calling a specific function on each element present in the parent array. It is a non-mutating method. Generally map() method is used to iterate over an array and calling function on every element of array.

If you want to iterate each value of the array and want to transform based of the logic,

we can use map function.

* map() creates a new array from calling a function for every array element.
* map() calls a function once for each element in an array.
* map() does not execute the function for empty elements.
* map() does not change the original array.

|  |
| --- |
| **Exmple** |
| var el = document.getElementById('root');  var arr = [2, 5, 6, 3, 8, 9];  var newArr = arr.map(function(val){  return (val\*val);  })  el.innerHTML = JSON.stringify(newArr); |
| //output - [4,25,36,9,64,81] |

1. **filter() method**

The arr.filter() method is used to create a new array from a given array consisting of only those elements from the given array which satisfy a condition set by the argument method.

* The filter() method creates a new array filled with elements that pass a test provided by a function.
* The filter() method does not execute the function for empty elements.
* The filter() method does not change the original array.

|  |
| --- |
| **Example** |
| function canVote(age) {  return age >= 18;  }  function func() {  var filtered = [24, 33, 16, 40].filter(canVote);  document.write(filtered);}  func(); |
| //output - 24,33,40 |

1. **Slice() method**

The arr.slice() method returns a new array containing a portion of the array on which it is implemented. The original remains unchanged.

* The slice() method returns selected elements in an array, as a new array.
* The slice() method selects from a given start, up to a (not inclusive) given end.
* The slice() method does not change the original array.

|  |
| --- |
| **Example** |
| var arr = [23,56,87,32,75,13];  var new\_arr = arr.slice(2,4);  document.write(arr);  document.write(new\_arr); |
| //output - [23,56,87,32,75,13]  [87,32] |

**Type Array (Returns single value):**

1. **Find index()**

The Array.findIndex() method is used to return the first index of the element in a given array that satisfies the provided testing function (passed in by user while calling). Otherwise, if no data is found then value of -1 is returned.

* The findIndex() method executes a function for each array element.
* The findIndex() method returns the index (position) of the first element that passes a test.
* The findIndex() method returns -1 if no match is found.
* The findIndex() method does not execute the function for empty array elements.
* The findIndex() method does not change the original array.

|  |
| --- |
| **Example** |
| // input array contain some elements.  var array = [-10, -0.20, 0.30, -40, -50];  // function (return element > 0).  var found = array.findIndex(function (element) {  return element > 0;  });  // Printing desired values.  document.write(found); |
| // output: 2 |

1. **IndexOf ()**

The arr.indexOf() method is used to find the index of the first occurrence of the search element provided as the argument to the method.

* The indexOf() method returns the first index (position) of a specified value.
* The indexOf() method returns -1 if the value is not found.
* The indexOf() method starts at a specified index and searches from left to right.
* By default the search starts at the first element and ends at the last.
* Negative start values counts from the last element (but still searches from left to right).

|  |
| --- |
| **Example** |
| function func() {  // Original string  var str = 'Departed Train';  // Finding index of occurrence of 'Train'  var index = str.indexOf('Train');  document.write(index);  } |
| // output: 9 |

1. **Find ()**

The arr.find() method in Javascript is used to get the value of the first element in the array that satisfies the provided condition. It checks all the elements of the array and whichever first element satisfies the condition is going to print. This function will not work function having the empty array elements, also does not change the original array.

* The find() method returns the value of the first element that passes a test.
* The find() method executes a function for each array element.
* The find() method returns undefined if no elements are found.
* The find() method does not execute the function for empty elements.
* The find() method does not change the original array.

|  |
| --- |
| **Example** |
| var array = [-10, -0.20, 0.30, -40, -50];  // Method (return element > 0).  var found = array.find(function (element) {  return element > 0;  });  // Printing desired values.  document.write(found); |
| // output: 0.30 |

1. **Reduce ()**

The arr.reduce() method in JavaScript is used to reduce the array to a single value and executes a provided function for each value of the array (from left-to-right) and the return value of the function is stored in an accumulator.

* The reduce() method executes a reducer function for array element.
* The reduce() method returns a single value: the function's accumulated result.
* The reduce() method does not execute the function for empty array elements.
* The reduce() method does not change the original array.

|  |
| --- |
| **Example** |
| const numbers = [175, 50, 25];  console.log(numbers.reduce(myFunc))  function myFunc(total, num) {  console.log(total,num)  return total + num;  } |
| // output: 100 |

**----------------------------------------OBJECT------------------------------------------------**

**Object:**

A javaScript object is an entity having state and behavior (properties and method).

javaScript object is collection of named values.

* JavaScript is an object-based language. Everything is an object in JavaScript.
* JavaScript is template based not class based. Here, we don't create class to get the object. But we direct create objects.

Creating Objects in JavaScript there are 3 ways to create objects.

1. By object literal

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

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

1. **By object literal:** The name: values pairs in JavaScript objects are called properties

* This is the easiest way to create a JavaScript Object.
* Using an object literal, you both define and create an object in one statement.
* An object literal is a list of name:value pairs (like age:50) inside curly braces {}.

|  |
| --- |
| const person = {  firstName: "John",  lastName: "Doe",  age: 50,  eyeColor: "blue"  }; |

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

The following example create a new JavaScript object using new Object(), and then adds 4 properties:

|  |
| --- |
| var emp = new Object();  emp.id = 1;  emp.name = "Komal";  emp.sal = 100000; |

1. **By using an object constructor (using new keyword)**

|  |
| --- |
| function emp (id,name,sal){  this.id=id;  this.name=name;  this.sal=sal;  }  e = new emp (10,"Yogesh",20); |

- How to Add, update and delete object preparties.

|  |
| --- |
| **Example** |
| const person= {  firstName:"John",  lastName:"Doe",  age:50,  22:"222",  eyeColor:"blue"  };  // To update value  person.age=22;  // To add property  person.location = "pune"  // To delete property  delete person.eyeColor;  console.log(person); |