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-----------------------------------**

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 |

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 |

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] |