|  |  |  |
| --- | --- | --- |
| Number Methods | | |
| 1 | toString ( ) |  |
| 2 | valueOf( ) |  |
| 3 | toLocalString( ) |  |
| 4 | toPrecision( ) |  |
| 5 | toFixed( ) |  |
| 6 | toExponential( ) |  |
| 7 | isInteger( ) |  |
| 8 | isNaN( ) |  |
| 9 | isFinite( ) |  |
|  |  |  |
|  |  |  |
| JSON | | |
| 1 | parse( ) |  |
| 2 | strinigify( ) |  |
|  |  |  |
| Boolean Methods | | |
| 1 | toString( ) |  |
| 2 | valueOf( ) |  |
| 3 | toSource( ) |  |
|  |  |  |
|  |  |  |
| String Methods | | |
| 1 | concat( ) |  |
| 2 | length( ) |  |
| 3 | toString( ) |  |
| 4 | valueOf( ) |  |
| 5 | toLowerCase( ) |  |
| 6 | toUpperCase( ) |  |
| 7 | toLocalLowerCase( ) |  |
| 8 | toLocalUpperCase( ) |  |
| 9 | indexOf( ) |  |
| 10 | lastIndexOf( ) |  |
| 11 | split( ) |  |
| 12 | slice( ) |  |
| 13 | match( ) |  |
| 14 | replace( ) |  |
| 15 | search( ) |  |
| 16 | substr( ) |  |
| 17 | substring( ) |  |
| 18 | charAt( ) |  |
| 19 | charCodeAt( ) |  |
| 20 | localeCompare( ) |  |
| 21 | trim( ) |  |
| 22 | startsWith( ) |  |
| 23 | endsWith( ) |  |
| 24 | repeat( ) |  |
|  |  |  |
| Array Methods | | |
| 1 | concat( ) |  |
|  | It joins 2 or more arrays  const arr1 = [1, 2, 3]  const arr2 = [4, 5, 6]  const arr3 = [7, 8]  const newarr = arr1.concat(arr2, arr3)  or  const newarr = [ ].concat(arr1, arr2, arr3) |  |
| 2 | join( ) |  |
|  | array.join( ‘*seperator’* )  It returns the array as string. Elements will be seperated by a specific seperator. Default seperator is comma  const fruits = [“Apple”, “Orange”, “Mango”]  const newFruit = fruits.join( ) // “Apple,Orange,Mango”  const newFruit = fruit.join(‘ ‘) // “Apple Orange Mang”  const newFruit = fruit.join(‘, ‘) // “Apple, Orange, Mango”  const newFruit = fruit.join(‘ - ‘) // “Apple-Orange-Mango” |  |
| 3 | toString( ) |  |
|  | toString works exacly same as join method but there are two difference   * toString menthod is not only for array but for all objects and * toString method separate array elements by comma only while using join we can define any separator   const fruits = [“Apple”, “Orange”, “Mango”]  const newFruit = fruits.toString( ) // “Apple,Orange,Mango” |  |
| 4 | push( ) |  |
|  | Adds new element or elements at last of array  array.push(item1, item2, item3…….)  const fruits = [“Apple”, “Orange”, “Mango”]  fruits.push(“Banana”) // 3  fruit // [“Apple”, “Orange”, “Mango”, “Banana”]  or  fruits.push(“Banana”, “Cherry”) // 3  fruit // [“Apple”, “Orange”, “Mango”, “Banana”, “Cherry] |  |
| 5 | pop( ) |  |
|  | Removes last element from an array and return that array. This changes the length of array  array.pop( )  const fruits = [“Apple”, “Orange”, “Mango”]  fruits.pop( ) // “Mango”  fruits // [“Apple”, “Orange”] |  |
| 6 | shift( ) |  |
|  | works same as pop but remove from starting of array. This changes the length or original array  array.shift( )  const fruits = [“Apple”, “Orange”, “Mango”]  fruits.shift() // “Apple”  fruits // [“Orange”, “Mango”] |  |
| 7 | unshift( ) |  |
|  | Works same as push but add at starting of array and returns new length of array  const fruits = [“Apple”, “Orange”, “Mango”]  fruits.unshift(“Banana”) // 3  fruit // [“Banana”, “Apple”, “Orange”, “Mango”]  or  fruits.push(“Banana”, “Cherry”) // 4  fruit // [“Banana”, “Cherry”, “Apple”, “Orange”, “Mango”]  const newArr = fruit.unshift(“Banana”)  newArr // 3 (new length or array) |  |
| 8 | slice( ) |  |
|  | array.slice( *start, end* )  This method returns the selected elements in an array, as a new array.  This selects the element includes at the given start argument but does not include the given end argument. This does not change original array  const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  const citrus = fruits.slice(1, 3); // [“Orange”, “Lemon”]  if no argument passed this will return full array  fruits.slice() //["Banana", "Orange", "Lemon", "Apple", "Mango"]  If 1 argument passed this will be considers start point and end point would be actual end of array  fruits.slice(1) //[ "Orange", "Lemon", "Apple", "Mango"]  fruits.slice(2) //[ "Lemon", "Apple", "Mango"]  If start and end would be same this would return empty array  fruits.slice(2, 2) // [ ]  If arguments would be negative count will start from end  fruits.slice(-3, -1) means start point is a from last and end point is 3 from last  // [“Lemon”, “Apple”] |  |
| 9 | splice( ) |  |
|  | The splice() method adds/removes items to/from an array, and returns the removed item(s). This changes original array  array.splice(starting index, howmany, item1, item2….)  const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  fruits.splice(2, 2) // [“Lemon”, “Apple”]  fruit // [“Banana”, “Orange”, “Mango”]  fruits.splice(2) // ["Lemon", "Apple", "Mango"]  fruit // [“Banana”, “Orange”]  fruits.splice(2, 2, “Guava”) // [“Lemon”, “Apple”]  fruit // [“Banana”, “Orange”, “Guava”, “Mango”] |  |
| 10 | sort( ) |  |
|  | This method sorts the item of an array. This changes the original array  const numbers = [ 30, 100, 2, 6, 35, 10]  numbers.sort( ) // [10, 100, 2, 30, 35, 6]  For Ascending order  numbers.sort( (a, b) => a-b) // [2, 6, 10, 30, 35, 100]  For Descending order  numbers.sort( (a, b) => b-a) // [100, 35, 30, 10, 6, 2] |  |
| 11 | reverse( ) |  |
|  | This method reverse the order of the element in an array |  |
| 12 | indexOf( ) |  |
|  | This method searches an array for the specific item. In case of same item at multiple position. it returns first item index only  array.indexOf(item)  It returns position(index) of item in array. If item not found in array it returns -1  const numbers = [1,2,3,1,4]  numbers.indexOf(1) // 0  numbers.indexOf(7) // -1 |  |
| 13 | lastIndexOf( ) |  |
|  | It method searches an array from last and returns postion of first occurance of that item(from last). It serches item from last but position(index) is counted from start only  const numbers = [1, 2, 3, 1, 4, 5]  numbers.lastIndexOf(1) // 3  numbers.lastIndexOf(7) // -1 |  |
| 14 | map( ) |  |
|  | const numbers = [ 10, 15, 20, 25]  const newNumbers = numbers.map( (item) => item \* 10 ) |  |
| 15 | filter( ) |  |
|  | This methods creates a new array with all array elements that pass a test ( provided as a function )  cons marks = [ 45, 29, 56, 65, 75]  const passed = marks.some( (item) => item >= 33 ) // [45, 56. 65. 75] |  |
| 16 | forEach( ) |  |
|  |  |  |
| 17 | reduce( ) |  |
|  |  |  |
| 18 | reduceRight( ) |  |
|  |  |  |
| 19 | some( ) |  |
|  | This method checks if any of the element in array pass a test (provided as a function).  If it finds an array element where the functiin returns true value, some() returns true and does not check remaining elements  cons marks = [ 45, 29, 56, 65, 75]  const isFail = marks.some( (item) => item < 33 ) |  |
| 20 | Every( ) |  |
|  | This method checks if all element in array pass a test (provided as a function).  If it finds an array element where the functiin returns false value, every() returns false and does not check remaining elements. If no false occur, this will return true.  cons marks = [ 45, 29, 56, 65, 75]  const isPass = marks.every( (item) => item >= 33 ) |  |
| 21 | find( ) |  |
|  | This method works exactly same as some( ) but only difference is that it returns first element value that passes test in place of true unline some().  This method checks if any of the element in array pass a test (provided as a function) and reuturns that element value.  If it finds an array element where the functiin returns true value, find() returns true and does not check remaining elements otherwise it returns undefined  cons marks = [ 45, 29, 56, 65, 75]  const isFail = marks.some( (item) => item < 33 ) |  |
| 22 | findIndex( ) |  |
|  | This method works exactly same as find() menthod, only difference is it returns the index of first element in place of value of element that passes the test |  |
| 23 | includes( ) |  |
|  | This method checks if an array contains a specific element. It returns true if the array contains the element, otherwise false  const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  const hasOrange = fruits.includes(“Orange”) // true |  |
| 24 | from( ) |  |
|  | This method returns an array from any object that have a length property or an iterable object  const newArray = Array.from(“ABCDE”) // [“A”, “B”, “C”, “D”, “E”] |  |
| 25 | flat( ) |  |
|  |  |  |
| 26 | flatMap( ) |  |
|  |  |  |
| 27 | fill( ) |  |
|  | This methods fill elements of an array with provided static value. Start and end postions can be defined. If not defined this will replace all elemets. start position will be included but last position will not be included. If last position is not defined this will replace all elemets starting from start position  array.fill(“value”, start, end)  const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  fruits.fill(“Kiwi”, 1, 3) = [“Orange”, “Kiwi”, “Kiwi”, “Kiwi”, “Mango”]  fruits.fill(“Kiwi”) = [“Kiwi”, “Kiwi”, “Kiwi”, “Kiwi”, “Kiwi”] |  |
| 28 | copyWithin( ) |  |
|  | This method copies array items to another position in the array, overwriting the existing values. This doen’t change length of array but it changes original array  array.copyWithin(target, startpos, endpos)  target is position starting where copied elements need to be replaced. startpos and endpos are positions of items being copied to place on another position.  If endpos is not provided this will copy single item only and if startpos is not provided this will copy starting from 0 index to last index means complete array  const x = [ 1, 2, 3, 4, 5, 6 ]  x.copyWithin( 2, 5) // [1, 2, **6**, 4, 5, 6]  x.copyWithin( 2, 4, 5) // [1, 2, **5**, **6**, 5, 6]  x.copyWithin( 2 ) // [1, 2, 1, 2, 3, 4] |  |
| 29 | entries( ) |  |
|  |  |  |
| 30 | isArray( ) |  |
|  | This method checks it an object is an array or not. It returns true if object is an array, otherwise it returns false |  |
| 31 | keys( ) |  |
|  |  |  |
| 32 | for loop |  |
|  | //for (initialize; check; increment) { execute }  for ( let i=0; i < arr.length; i++) {  console.log( arr[ i ]  }  for looping revese in array  for ( let i=arr.length-1; i >=0; i--) {  console.log( arr[ i ]  } |  |
| 33 | for in loop |  |
|  | // for ( somevar in object ) { execute }  for (let item in obj){  console.log(item)  console.log(obj[item]  console.log( “in” item “we have” obj[item]  } |  |
|  |  |  |
| Date Methods | | |  |
| 1 | Date( ) |  |
| 2 | getDate( ) |  |
| 3 | getDay( ) |  |
| 4 | getMonth( ) |  |
| 5 | getYear ( ) |  |
| 6 | getFullYear( ) |  |
| 7 | getHourse( ) |  |
| 8 | getMinutes( ) |  |
| 9 | getSeconds( ) |  |
| 10 | getMilliSeconds( ) |  |
| 11 | getTime( ) |  |
| 12 | getTimezoneOffset( ) |  |
| 13 | set…….. |  |
| 14 | toDateString( ) |  |
| 15 | toLocalDateString( ) |  |
| 16 | toString( ) |  |
| 17 | toTimeString( ) |  |
| 18 | valueOf( ) |  |
| 19 | Date.parse( ) |  |
|  |  |  |
| Math Object | | |
| 1 | random( ) |  |
| 2 | round( ) |  |
| 3 | min( ) |  |
| 4 | max( ) |  |
| 5 | floor( ) |  |
| 6 | ceil( ) |  |
| 7 | abs( ) |  |
|  |  |  |
| Statements | | |
| 1 | break |  |
| 2 | continue |  |
| 3 | return |  |
| 4 | switch |  |
| 5 | throw |  |
| 6 | try…catch…finally |  |
| 7 | while |  |
| 8 | do…while |  |
| 9 | for |  |
| 10 | for…in |  |
| 11 | for…of |  |
| 12 | if…else |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
|  |  |
| object.length | undefined |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  | Array.join & Array.toString() |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |