**Nested Arrays:**

const arr1 = [0, 1, 2, [3, 4]];

console.log(arr1.flat());

// output: [0, 1, 2, 3, 4]

const arr2 = [0, 1, 2, [[[3, 4]]]];

console.log(arr2.flat(2));

// output: [0, 1, 2, [3, 4]]

const arr6 = [1, 2, [3, 4, [5, 6, [7, 8, [9, 10]]]]];

console.log(arr6.flat(Infinity));

// [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

[**Splitting an array with slice()**](https://codepen.io/cferdinandi/pen/ExEGXdj) **:**

let wizards = ['Merlin', 'Gandalf', 'Ursula', 'Radagast'];

let wizardsFirst = wizards.slice(0, 2); //Merlin, Gandalf

let wizardsSecond = wizards.slice(2); //Ursula, Radagast

console.log(wizardsFirst, wizardsSecond);

**String.replace:**

let str="Obed Gnanam";

let newStr=str.replace("Gnanam", "Sundar");

console.log(newStr); // Obed Sundar

**Array replace/update:**

const arr = ["obed", "Stainz", "leela"];

arr[1] = "Sundar";

console.log(arr);

//Output: obed, Sundar, leela

**String Concatenation:**

let str1="Obed";

let Str2=" Gnanam";

let fullStr=str1.concat(Str2);

document.write(fullStr); // Obed Gnanam

**For Loop in Array:**

The JavaScript for loop is used to **iterate** through the array or **the elements for a specified number of times.**

**Example:**

const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];

let text = "";

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

text += cars[i] + "<br>";

}

document.write(text);

**Ex 2:**

for (let i = 0; i < 5; i++) {

console.log(i);

} // Output: 0, 1, 2, 3, 4

**For each Loop in Array:**

The forEach() method is also used to loop through arrays, but it uses a function differently than the classic “for loop”. It passes a **callback function** for each element of an array.

**Example:**

const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];

let text = "";

cars.forEach(myFunction);

function myFunction(i) {

text += i + "<br>";

}

document.write(text);

**Ex 2:**

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

numbers.forEach(number => {

console.log(number);

}); // Output: 1, 2, 3, 4, 5

**For each Loop in Set:**

const cars = new Set (["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"]);

let text = "";

cars.forEach(function(value){

text += value + "<br>";

})

document.write(text);

**For each Loop in Map:**

const cars = new Map([

["BMW", 500], ["Volvo", 300], ["Audi", 200]

]);

let text = "";

cars.forEach (function(value, key) {

text += key + " " + value + "<br>"

})

document.write(text);

**For of Loop:**

This for (..of) statement lets you loop over the data structures that are iterable such as **Arrays, Strings, Maps, Node Lists**, and more.

**Example:**

const cars = ["BMW", "Volvo", "Saab"];

let text = "";

var i;

for (i of cars) {

text += i + "<br>";

}

document.write(text);

**Ex 2:**

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

for (const number of numbers) {

console.log(number);

} // Output: 1, 2, 3, 4, 5

**For in Loop:**

The loop will iterate over all enumerable properties of the **object** itself.

**Example:**

const cars = {1:"BMW", 2:"Volvo", 3:"Saab"};

let text = "";

var i;

for (i in cars) {

text += cars[i] + "<br>";

}

document.write(text);

**While Loop:**

Loops a code block while a **condition is true**.

let text="";

var i=0;

while(i<10){

text+= i + "<br>" ;

i++;

}

document.write(text);

**Do While Loop:**

Loops a code block **once**, and then **while a condition is true.**

let text="";

var i=0;

do{

text+= i + "<br>" ;

i++;

}

while(i<10)

document.write(text);

**Constructor Function:**

function Person(first, last, age) {

this.firstName = first;

this.lastName = last;

this.age = age;

}

const myFather=new Person("Sundar", "Muthusamy", "54");

//const myMother=new Person("Leela", "Sundar", "54");

document.write(+myFather.age);

// output:54

**setInterval** and **setTimeout**

setInterval - Repeatedly executes a function or a piece of code at specified intervals.

setTimeout - Executes a function or a piece of code once after a specified delay.

**Repetition:**

**setInterval** repeats the execution of the provided function at regular intervals.

**setTimeout** executes the provided function once after a specified delay.

**Cancellation:**

Both can be canceled using their respective **clearInterval** and **clearTimeout** functions.

let counter = 0;

const intervalId = setInterval(() => {

  console.log(`This code runs every 1000 milliseconds. Counter: ${counter}`);

  counter++;

  if (counter === 5) {

    console.log('Clearing the interval after 5 executions.');

    clearInterval(intervalId);

    // Delayed execution after clearing the interval

    setTimeout(() => {

      console.log('This code runs once after clearing the interval.');

    }, 2000);

  }

}, 1000);

**Fibonacci Series Up to n Terms (For Loop)**

// program to generate fibonacci series up to n terms

// take input from the user

const number = parseInt(prompt('Enter the number of terms: '));

let n1 = 0, n2 = 1, nextTerm;

for (let i = 1; i <= number; i++) {

console.log(n1);

nextTerm = n1 + n2;

n1 = n2;

n2 = nextTerm;

}

## Fibonacci Sequence Up to a Certain Number

// program to generate fibonacci series up to a certain number

// take input from the user

const number = parseInt(prompt('Enter a positive number: '));

let n1 = 0, n2 = 1, nextTerm;

console.log('Fibonacci Series:');

console.log(n1); // print 0

console.log(n2); // print 1

nextTerm = n1 + n2;

while (nextTerm <= number) {

// print the next term

console.log(nextTerm);

}

n1 = n2;

n2 = nextTerm;

nextTerm = n1 + n2;

}

## Pass by Value in JavaScript

let num1 = 70

let num2 = num1

console.log(num1) *// 70*

console.log(num2) *// 70*

num1 = 40

console.log(num1) *// 40*

console.log(num2) *// 70*

## Pass by Reference in JavaScript

let obj1 = {website: "Scaler Academy"}

let obj2 = obj1;

console.log(obj1) *// {website: "Scaler Academy"}*

console.log(obj2) *// {website: "Scaler Academy"}*

obj1.website = "Scaler Topics"

console.log(obj1) *// {website: "Scaler Topics"}*

console.log(obj2) *// {website: "Scaler Topics"}*

**Closure Function:**

function outerFunction() {

  // Outer function's variable

  let outerVariable = "I am from the outer function";

  function innerFunction() {

    // Inner function has access to outer function's variable

    console.log(outerVariable);

  }

  // Return the inner function

  return innerFunction;

}

// Call the outer function and store the returned inner function

const closureExample = outerFunction();

// Call the inner function, which still has access to outerVariable

closureExample(); // Outputs: I am from the outer function

**Print the occurrence of the characters in a string:**

**Method 1:**

*//print occurrence of chars*

function print(ans){

for (let [key, value] of ans){

console.log(`${key} occurs ${value} times`);

}

}

*//count occurrence of char*

function count(str,let\_map){

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

let k= let\_map.get(str[i]);

let\_map.set(str[i], k+1);

}

print(let\_map);

}

*//map*

function count\_occurs(test, callback){

if(test.length === 0){

console.log("empty");

return;

}

else {

let ans = new Map();

for (let i=0; i<test.length; i++)

{

ans.set(test[i], 0);

}

callback (test, ans);

}

}

let test = "javascript is programming";

count\_occurs ( test, count);

**Method 2:**

function count\_occur(str){

if (str.length == 0){

console.log("invalid")

return;

}

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

let count = 0;

for (let j=0; j<str.length; j++){

if(str[i] == str[j] && i>j)

{

break;

}

if(str[i] == str[j]){

count++;

}

}

if (count > 0)

console.log(`${str[i]} occurs ${count} times`);

}

}

let test\_str = "javascript is programming";

count\_occur(test\_str);

**Method 3: Easy way to find one character:**

function test(){

let str= "javascript is programming";

*//let j= (str.match(/j/g)}.length;*

console.log(

*//(str.match(/j/g)).length);*

(str.match(/a/g)).length);

*//(str.match(/v/g)).length);*

}

test();

**Replace Even Letters with Dash:**

function replaceEvenLettersWithDash(inputString) {

if (typeof inputString !== 'string') {

return "Invalid input. Please provide a string.";

}

let result = '';

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

if (i % 2 === 0) {

result += '-';

} else {

result += inputString[i];

}

}

return result;

}

*// Test the function with an example string*

const inputString = "abcdefghijklmnopqrstuvwxyz";

const output = replaceEvenLettersWithDash(inputString);

console.log(output);

**Check Palindrome:**

function isPalindrome(str) {

  // Remove spaces and convert to lowercase for case-insensitive comparison

  const cleanStr = str.replace(/\s/g, "").toLowerCase();

  // Reverse the string

  const reversedStr = cleanStr.split("").reverse().join("");

  // Compare the original and reversed strings

  return cleanStr === reversedStr;

}

// Test case

console.log(isPalindrome("madam")); // true

**Check Anagram:**

function areAnagrams(str1, str2) {

*// Remove spaces and convert to lowercase*

str1 = str1.replace(/\s/g, '').toLowerCase();

str2 = str2.replace(/\s/g, '').toLowerCase();

*// Sort the characters in both strings and compare*

const sortedStr1 = str1.split('').sort().join('');

const sortedStr2 = str2.split('').sort().join('');

return sortedStr1 === sortedStr2;

}

*// Example usage:*

const str1 = "listen";

const str2 = "silent";

console.log(areAnagrams(str1, str2)); *// Output: true*

**Ascending / Sorting an Array of data by id or age:**

We can sort the array by id or age.

const users = [

{ id: 1, name: "shankar", isActive: true, age: 55 },

{ id: 2, name: "raju", isActive: true, age: 15 },

{ id: 3, name: "pavan", isActive: false, age: 76 },

{ id: 4, name: "shankar", isActive: true, age: 55 },

{ id: 5, name: "ajay", isActive: false, age: 24 },

{ id: 6, name: "bhuvan", isActive: true, age: 77 }

];

const usersSortedAscending = users.slice().sort((a, b) => a.id - b.id);

// const usersSortedAscending = users.slice().sort((a, b) => a.age - b.age);

console.log(usersSortedAscending);

**Filter out just the names from the array of objects in JavaScript:**

const a = [{name: "Obed", age: 24}, {name: "Sundar", age: 32}];

const names = a.map(obj => obj.name);

console.log(names); // Output: ["Obed", "Sundar"]

**Inserting a value at a specific index in a JavaScript array**

let myArray = [1, 2, 3, 4, 5];

let index = 2; // Index at which you want to insert the value

let valueToInsert = 10;

myArray.splice(index, 0, valueToInsert);

console.log(myArray);

* **index** is the position at which the new element (**valueToInsert**) will be inserted.
* **0** indicates that no elements should be removed before inserting the new element. It means "insertion without deletion".

**Check and Print vowels:**

function countVowelsAndPrint(inputString) {

  // Count the number of vowels

  var vowelCount = inputString.match(/[aeiouAEIOU]/g).length;

  // Create a formatted string

  var resultString = `The number of vowels in "${inputString}" is: ${vowelCount}`;

  // Print the result

  console.log(resultString);

}

// Example usage:

var exampleString = "Hello World!";

countVowelsAndPrint(exampleString);

**Print the occurrences of the words in a sentence**

function printWordOccurrences(inputSentence) {

  // Remove punctuation and convert the sentence to lowercase

  var cleanedSentence = inputSentence.replace(/[.,\/#!$%\^&\\*;:{}=\-\_`~()]/g, "").toLowerCase();

  // Split the sentence into an array of words

  var wordsArray = cleanedSentence.split(' ');

  // Initialize an object to store word occurrences

  var wordOccurrences = {};

  // Loop through the array of words and update word occurrences

  for (var word of wordsArray) {

    if (wordOccurrences[word]) {

      wordOccurrences[word]++;

    } else {

      wordOccurrences[word] = 1;

    }

  }

  // Print the result

  for (var word in wordOccurrences) {

    console.log(`The word "${word}" appears ${wordOccurrences[word]} times in the sentence.`);

  }

}

// Example usage:

var exampleSentence = "Hello world! Hello JavaScript world!";

printWordOccurrences(exampleSentence);

**Find the Strings with Repeated Letters:**

function findStringsWithRepeatedLetters(strings) {

  // Array to store strings with repeated letters

  let result = [];

  // Function to check if a string has repeated letters

  function hasRepeatedLetters(str) {

    let charCount = {};

    for (let char of str) {

      if (charCount[char]) {

        // If the character already exists in the charCount object, it's repeated

        return true;

      } else {

        charCount[char] = 1;

      }

    }

    // No repeated letters found

    return false;

  }

  // Iterate through the array of strings

  for (let str of strings) {

    if (hasRepeatedLetters(str)) {

      result.push(str);

    }

  }

  return result;

}

// Example usage:

let strings = ["hello", "world", "apple", "banana"];

let result = findStringsWithRepeatedLetters(strings);

console.log("Strings with repeated letters:", result);

**Coding Challenge:**

(function() {

console.log(1);

setTimeout(function(){console.log(2)}, 1000);

setTimeout(function(){console.log(3)}, 0);

console.log(4);

})();

Output:

1

4

3

2

**Coding Challenge:**

for (var i = 0; i < 3; i++) {

setTimeout(function() { console.log(i); }, 1000 + i);

}

Output:

3

3

3

**Coding Challenge:**

for (var i = 0; i < 3; i++) {

(function (index) {

setTimeout(function() { console.log(index); }, 1000 + index);

})(i);

}

Output:

0

1

2

**To find the second Most Array element:**

function findSecondLargest(arr) {

  // Sort the array in descending order

  arr.sort((a, b) => b - a);

  // The second element in the sorted array is the second largest

  return arr[1];

}

// Example usage:

const numbers = [10, 5, 8, 20, 2];

const secondLargest = findSecondLargest(numbers);

console.log("Second Largest Element:", secondLargest);

Output :

10

let a = 4;

console.log(this.a); // Outputs: undefined

In this case, the output of **this.a** would be **undefined**. This is because the variable **a** is declared using **let** in the global scope, and it is not added as a property to the global object (**this**).

var a = 4;

console.log(this.a); // Outputs: 4

In this case, the output of **this.a** would be **4** because **a** is a property of the global object (**this**), and **this.a** refers to the value of that property.

for(var i=0;i<5;i++){

setTimeout(()=>console.log(i), 5000);

}

Output: 5

5

5

5

5

for(let i=0;i<5;i++){

setTimeout(()=>console.log(i), 5000);

}

Output:

0

1

2

3

4

const test = [

  { name: 'test name1', isStudent: true },

  { name: 'test name2', isStudent: false },

  { name: 'test name3', isStudent: true }

];

let height = 0;

console.log(height || 100); // Using || (logical OR) operator

console.log(height ? ? 100); // Using ?? (nullish coalescing) operator

output:

100

0

let user = {};

  1. console.log(user.address.street); // Output: undefined

  2. console.log(user?.address?.street); // Output: undefined

function fun(a, ...b, c) any issue?

//Uncaught SyntaxError: Rest parameter must be last formal parameter"

let a = "1", b=2, c=3 then what will be the value of a+b+c?

// Output: “123”

const test1 = { name: 'test name1', isStudent: true };

const test2 = { name: 'test name2', isStudent: false };

// Updated object with nested objects

const updatedObject1 = { test1, test2 };

// Updated object with spread operator

const updatedObject2 = { ...test1, ...test2 };

console.log(updatedObject1);

console.log(updatedObject2);

output:

// updatedObject1

{

  test1: { name: 'test name1', isStudent: true },

  test2: { name: 'test name2', isStudent: false }

}

// updatedObject2

{

  name: 'test name2',

  isStudent: false

}

Explanation:

* **updatedObject1** is an object containing two properties, **test1** and **test2**, each of which holds the corresponding objects.
* **updatedObject2** is created by using the spread operator (**...**) on **test1** and **test2**. It merges the properties of **test1** and **test2** into a new object. If there are conflicting keys, the values from the second object (**test2** in this case) will override the values from the first object (**test1**).

const test = [

  { name: 'test name1', isStudent: true },

  { name: 'test name2', isStudent: false },

  { name: 'test name3', isStudent: true }

];

// Filter objects with isStudent: true and extract names

const namesWithIsStudentTrue = test

  .filter(obj => obj.isStudent)

  .map(obj => obj.name);

console.log(namesWithIsStudentTrue);

output:

['test name1', 'test name3']