1. Do the below programs in anonymous function & IIFE
   1. **Print odd numbers in an array**

Anonymous Function:

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

const printOddNumbers = function(arr) {

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

if (arr[i] % 2 !== 0) {

console.log(arr[i]);

}

}

};

printOddNumbers(numbers);

In this example, we define an anonymous function printOddNumbers that takes an array as an argument and iterates through it, printing the odd numbers. We then call the function with the numbers array.

IFE (Immediately Invoked Function Expression):

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

(function(arr) {

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

if (arr[i] % 2 !== 0) {

console.log(arr[i]);

}

}

})(numbers);

In this example, we define an IIFE that takes the numbers array as its argument and immediately invokes the function, printing the odd numbers.

Both of these approaches will print the odd numbers in the numbers array.

**b. Convert all the strings to title caps in a string array**

Anonymous Function:

const stringArray = ["hello world", "good morning", "javascript is fun"];

function convertToTitleCaps(arr) {

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

const words = arr[i].split(' ');

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

words[j] = words[j][0].toUpperCase() + words[j].slice(1);

}

arr[i] = words.join(' ');

}

}

convertToTitleCaps(stringArray);

console.log(stringArray);

In this example, we define an anonymous function convertToTitleCaps that takes an array of strings and iterates through it, converting each string to title case. The function splits each string into words, capitalizes the first letter of each word, and then joins them back together. We then call the function with the stringArray and print the modified array.

IIFE (Immediately Invoked Function Expression):

const stringArray = ["hello world", "good morning", "javascript is fun"];

(function(arr) {

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

const words = arr[i].split(' ');

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

words[j] = words[j][0].toUpperCase() + words[j].slice(1);

}

arr[i] = words.join(' ');

}

})(stringArray);

console.log(stringArray);

In this example, we define an IIFE that takes the stringArray as its argument and immediately invokes the function, converting each string to title case. The modified array is then printed.

Both of these approaches will convert all the strings in the stringArray to title case.

**C. Sum of all numbers in an array**

Anonymous Function:

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

const sum = function(arr) {

let result = 0;

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

result += arr[i];

}

return result;

};

const totalSum = sum(numbers);

console.log(totalSum); // Output will be 15 (1 + 2 + 3 + 4 + 5)

IIFE (Immediately Invoked Function Expression):

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

const totalSum = (function(arr) {

let result = 0;

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

result += arr[i];

}

return result;

})(numbers);

console.log(totalSum); // Output will be 15 (1 + 2 + 3 + 4 + 5)

In the first example, we define an anonymous function sum and then call it with the numbers array to calculate the sum. In the second example, we create an IIFE that immediately calculates the sum of the numbers in the numbers array and assigns it to the totalSum variable. Both approaches will give you the same result.

**d. Return all the prime numbers in an array**

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

const isPrime = function(num) {

if (num <= 1) return false;

if (num <= 3) return true;

if (num % 2 === 0 || num % 3 === 0) return false;

for (let i = 5; i \* i <= num; i += 6) {

if (num % i === 0 || num % (i + 2) === 0) return false;

}

return true;

};

const primeNumbers = numbers.filter(isPrime);

console.log(primeNumbers); // Output will be [2, 3, 5, 7]

IIFE (Immediately Invoked Function Expression):

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

const primeNumbers = (function(arr) {

const isPrime = function(num) {

if (num <= 1) return false;

if (num <= 3) return true;

if (num % 2 === 0 || num % 3 === 0) return false;

for (let i = 5; i \* i <= num; i += 6) {

if (num % i === 0 || num % (i + 2) === 0) return false;

}

return true;

};

return arr.filter(isPrime);

})(numbers);

console.log(primeNumbers); // Output will be [2, 3, 5, 7]

In both examples, we define a function isPrime to check whether a number is prime or not. Then, we either use filter to create a new array containing only the prime numbers or return the result directly in the IIFE. The prime numbers in the numbers array are then stored in the primeNumbers variable.

**d. Return all the palindromes in an array**

Anonymous Function:

const words = ["level", "hello", "rotor", "world", "madam"];

const isPalindrome = function(word) {

const reversedWord = word.split('').reverse().join('');

return word === reversedWord;

};

const palindromeWords = words.filter(palindrome);

console.log(palindromeWords); // Output will be ["level", "rotor", "madam"]

IIFE (Immediately Invoked Function Expression):

const words = ["level", "hello", "rotor", "world", "madam"];

const palindromeWords = (function(arr) {

const isPalindrome = function(word) {

const reversedWord = word.split('').reverse().join('');

return word === reversedWord;

};

return arr.filter(isPalindrome);

})(words);

console.log(palindromeWords); // Output will be ["level", "rotor", "madam"]

In both examples, we define a function isPalindrome to check whether a word is a palindrome or not. Then, we either use filter to create a new array containing only the palindromic words or return the result directly in the IIFE. The palindromic words in the words array are then stored in the palindromeWords variable.

**e. Return median of two sorted arrays of the same size.**

const arr1 = [1, 3, 5];

const arr2 = [2, 4, 6];

const findMedian = function(nums1, nums2) {

const merged = nums1.concat(nums2);

const sorted = merged.sort((a, b) => a - b);

const length = sorted.length;

if (length % 2 === 0) {

const mid = length / 2;

return (sorted[mid - 1] + sorted[mid]) / 2;

} else {

const mid = Math.floor(length / 2);

return sorted[mid];

}

};

const median = findMedian(arr1, arr2);

console.log(median); // Output will be 3.5 (average of 3 and 4)

IIFE (Immediately Invoked Function Expression):

const arr1 = [1, 3, 5];

const arr2 = [2, 4, 6];

const median = (function(nums1, nums2) {

const merged = nums1.concat(nums2);

const sorted = merged.sort((a, b) => a - b);

const length = sorted.length;

if (length % 2 === 0) {

const mid = length / 2;

return (sorted[mid - 1] + sorted[mid]) / 2;

} else {

const mid = Math.floor(length / 2);

return sorted[mid];

}

})(arr1, arr2);

console.log(median); // Output will be 3.5 (average of 3 and 4)

In both examples, we merge the two arrays, sort the merged array, and then calculate the median based on whether the length is even or odd. The median is then either returned directly or stored in the median variable.

**g. Remove duplicates from an array**

Anonymous Function:

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

const removeDuplicates = function(arr) {

return arr.filter((value, index, self) => {

return self.indexOf(value) === index;

});

};

const uniqueNumbers = removeDuplicates(numbers);

console.log(uniqueNumbers); // Output will be [1, 2, 3, 4, 5]

IIFE (Immediately Invoked Function Expression):

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

const uniqueNumbers = (function(arr) {

return arr.filter((value, index, self) => {

return self.indexOf(value) === index;

});

})(numbers);

console.log(uniqueNumbers); // Output will be [1, 2, 3, 4, 5]

In both examples, we define a function removeDuplicates that uses the filter method to create a new array containing only the unique values. We check whether the index of the current value is equal to its first occurrence in the array to determine uniqueness. The unique values are then either returned directly or stored in the uniqueNumbers variable.

**h. Rotate an array by k times**

Anonymous Function:

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

const k = 2;

const rotateArray = function(arr, k) {

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

const lastItem = arr.pop();

arr.unshift(lastItem);

}

return arr;

};

const rotatedNumbers = rotateArray(numbers, k);

console.log(rotatedNumbers); // Output will be [4, 5, 1, 2, 3]

IIFE (Immediately Invoked Function Expression):

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

const k = 2;

const rotatedNumbers = (function(arr, k) {

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

const lastItem = arr.pop();

arr.unshift(lastItem);

}

return arr;

})(numbers, k);

console.log(rotatedNumbers); // Output will be [4, 5, 1, 2, 3]

In both examples, we define a function rotateArray (or use the logic directly in the IIFE) that rotates the array by shifting elements from the end to the beginning k times. The rotated array is then either returned directly or stored in the rotatedNumbers variable

1. Do the below programs in arrow functions.

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

const printOddNumbers = (arr) => {

arr.forEach((number) => {

if (number % 2 !== 0) {

console.log(number);

}

});

};

printOddNumbers(numbers);

In this code, we define an arrow function printOddNumbers that takes an array as its argument. Inside the function, we use the forEach method to iterate through each element in the array. For each element, we check if it's an odd number (i.e., number % 2 !== 0), and if it is, we print it to the console using console.log.

**b. Convert all the strings to title caps in a string array**

const stringArray = ["hello world", "this is a test", "programming is fun"];

const convertToTitleCase = (str) => {

return str

.split(" ")

.map((word) => word.charAt(0).toUpperCase() + word.slice(1))

.join(" ");

};

const titleCaseArray = stringArray.map(convertToTitleCase);

console.log(titleCaseArray);

In this code:

We define an arrow function convertToTitleCase that takes a string str as its argument.

Inside the convertToTitleCase function, we split the input string into an array of words using .split(" ").

Then, we use the map method to iterate through each word in the array and convert the first character of each word to uppercase using word.charAt(0).toUpperCase(), and we concatenate it with the rest of the word using word.slice(1).

Finally, we join the array of words back into a string with spaces between them using .join(" ").

We then use the map method to apply the convertToTitleCase function to each element in the stringArray, resulting in a new array called titleCaseArray where all strings are in title case.

**c. Sum of all numbers in an array**

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

const sumOfNumbers = (arr) => {

return arr.reduce((total, number) => total + number, 0);

};

const result = sumOfNumbers(numbers);

console.log(result);

You can calculate the sum of all numbers in an array using an arrow function and the reduce method. Here's an example:

javascript

Copy code

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

const sumOfNumbers = (arr) => {

return arr.reduce((total, number) => total + number, 0);

};

const result = sumOfNumbers(numbers);

console.log(result);

In this code:

We define an arrow function sumOfNumbers that takes an array arr as its argument.

Inside the sumOfNumbers function, we use the reduce method to iterate through each element in the array. The reduce method takes two parameters: a callback function and an initial value (in this case, 0). The callback function takes two arguments, total (which accumulates the sum) and number (the current element being processed).

In the callback function, we add the number to the total. This process continues for each element in the array, accumulating the sum.

Finally, the reduce method returns the total sum assigned to the result variable and then printed to the console.

This code will calculate and print the sum of all numbers in the numbers array using an arrow function and the reduce method.

**d. Return all the prime numbers in an array**

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

const isPrime = (number) => {

if (number <= 1) return false;

if (number <= 3) return true;

if (number % 2 === 0 || number % 3 === 0) return false;

for (let i = 5; i \* i <= number; i += 6) {

if (number % i === 0 || number % (i + 2) === 0) return false;

}

return true;

};

const primeNumbers = numbers.filter((number) => isPrime(number));

console.log(primeNumbers);

In this code:

We define an arrow function isPrime that takes a number as its argument. This function checks if the number is prime using a common prime-checking algorithm.

Inside the isPrime function, we first handle some edge cases:

If the number is less than or equal to 1, it's not prime, so we return false.

If the number is 2 or 3, it's prime, so we return true.

For other numbers greater than 3, we use a loop to check if they are divisible by any number from 5 to the square root of the number. If they are divisible, we return false, indicating that the number is not prime. Otherwise, we return true.

We define an arrow function that uses the filter method to iterate through the numbers array and keep only those numbers for which isPrime returns true.

The resulting array, primeNumbers, will contain all the prime numbers from the original numbers array, and we print it to the console.

This code will return and print all the prime numbers in the numbers array using an arrow function and the filter method.

**e. Return all the palindromes in an array**

const strings = ["level", "hello", "radar", "world", "deified"];

const isPalindrome = (str) => {

// Remove non-alphanumeric characters and convert to lowercase

const cleanedStr = str.replace(/[^a-zA-Z0-9]/g, "").toLowerCase();

// Check if the cleaned string is the same when reversed

return cleanedStr === cleanedStr.split("").reverse().join("");

};

const palindromes = strings.filter((str) => isPalindrome(str));

console.log(palindromes);

In this code:

We define an arrow function isPalindrome that takes a string str as its argument. This function checks if the string is a palindrome.

Inside the isPalindrome function, we clean the input string by removing non-alphanumeric characters and converting it to lowercase. This step ensures that we can correctly identify palindromes regardless of capitalization or special characters.

We then check if the cleaned string is the same when reversed. If it is, we return true, indicating that the string is a palindrome. Otherwise, we return false.

We define an arrow function that uses the filter method to iterate through the strings array and keep only those strings for which isPalindrome returns true.

The resulting array, palindromes, will contain all the palindromes from the original strings array, and we print it to the console.

This code will return and print all the palindromes in the strings array using arrow functions and the filter method.