2804. Array Prototype ForEach

Easy

Problem Description

arrays within that context.

method is a built-in function that executes a provided function once for each array element. However, this custom method should be written without using the native for Each or other built-in array methods. The custom forEach method needs to accept two parameters:

The problem requires the creation of a custom for Each method that can be applied to all arrays. In JavaScript, the for Each

context: an object that can be referred to as this within the scope of the callback function.

The callback function itself should have access to three arguments:

• callback: a function that you want to execute for each element in the array.

2. index: the index of the current array element. 3. array: the entire array that the forEach is being executed upon.

- 1. currentValue: the value of the array element in the current iteration.

The essence of this problem is to enhance the array prototype inside JavaScript, ensuring that this functionality is available on all

Here is the step-by-step breakdown of the approach:

Intuition The intuition is to replicate the behavior of the built-in for Each functionality manually. We want to achieve three key tasks:

2. During each iteration, execute the callback function with the proper arguments – we can call the callback function with currentValue, index,

1. Iterate through each element of the array – we can do this via a for loop that starts at the beginning of the array (index 0) and continues to the end (length of the array).

and array.

3. Respect the context in which callback is executed if provided – we can use the call method on the callback function to set the this value explicitly to the context passed as a parameter.

Solution Approach

- The implementation of the custom for Each method makes use of JavaScript's prototypal inheritance. By appending a new
- function to Array.prototype, we ensure that all arrays inherit this method.

We start by adding a new function to Array prototype named for Each. This means that every array created in this JavaScript environment will now have access to this custom for Each method.

method was called.

or to undefined in strict mode.

array is ['Alice', 'Bob', 'Charlie'].

function greeting(name, index) {

names.forEach(greeting);

const names = ['Alice', 'Bob', 'Charlie'];

context, so it defaults to the global context.

The final output to the console would be:

Hello, Alice! You are at position 1.

Hello, Charlie! You are at position 3.

def forEach(self, callback, context=None):

Example usage of the custom forEach function

Create an instance of Array with numbers

arr.forEach(double_values_callback, context)

print(arr) # Output will be: [2, 4, 6]

import java.util.function.BiConsumer;

Iterate over each element in the list

callback(value, index, self)

def double values callback(value, index, array, context=None):

Update the list element to its doubled value

interface ForEach<T> extends BiConsumer<T, Integer> {

for index, value in enumerate(self):

function for each element in an array without relying on the native method.

The custom forEach function added to the Array class (inherits from list)

If context is provided, apply it to the callback

Define a callback function that doubles the value of each list element

Print the updated list to the console, which should show doubled values

// Custom interface extending the functionality of BiConsumer interface

* A custom implementation of the forEach function that operates on arrays.

The type of the elements in the array.

* @param callback The callback function to execute for each element.

public static <T> void forEach(T[] array. ForEach<T> callback) {

ForEach<Integer> doubleValuesCallback = new ForEach<>() {

public void accept(Integer value, Integer index) {

System.out.println(value); // Output will be: 2, 4, 6

// Iterate over each element in the array

for (int i = 0; i < array.length; <math>i++) {

callback.accept(array[i], i);

arr[index] = value * 2;

ArrayUtils.forEach(arr, doubleValuesCallback);

// Create an array of numbers

Integer[] arr = $\{1, 2, 3\}$;

for (Integer value : arr) {

// Iterate over each element in the vector

// Example usage of the custom forEach function

// Double the passed element's value

for (size t i = 0; i < vec.size(); ++i){

callback(vec[i], i, vec);

// Create a vector of numbers

value *= 2;

std::vector<int> arr = {1, 2, 3};

The array on which the operation is performed.

// Execute the callback function with the current element and its index

// Define a callback function that doubles the value of each array element

// Print the updated array to the console, which should show doubled values

// Apply the custom forEach function to the arr array using the doubleValuesCallback

void forEach(std::vector<T>& vec. std::function<void(T&, size_t, std::vector<T>&)> callback, void* context = nullptr) {

// Call the callback function, passing the current element by reference, its index, and the vector itself

Apply the custom forEach function to the arr Array using the doubleValuesCallback and context

Hello, Bob! You are at position 2.

array itself (this). This matches the standard forEach method's signature.

Here's a breakdown of how we could use our custom for Each method to accomplish this:

console.log(`Hello, \${name}! You are at position \${index + 1}.`);

each element of the array. The context is optional and is used to specify the value of this inside the callback function when it is called.

Our forEach function takes in two parameters: callback and context. The callback is a function that we want to call with

- We make use of a simple for loop to iterate over the array. The initial index is set to 0, and we loop through until we reach the end of the array, denoted by this length, because within this function, this refers to the array upon which the for Each
- For each iteration, we use the call method of the callback function to execute it. The call method is a built-in JavaScript method that allows us to call a function with an explicitly set this value - in this case, the context parameter. If context is

undefined, the value of this inside the callback will default to the global object, which is the default behavior in a browser,

The callback function is called with three arguments: the current element value (this[i]), the current index (i), and the

The code does not include any complex algorithms, data structures, or patterns—it is a straightforward iteration using a for loop

The beauty of this approach is in its simplicity and direct manipulation of the Array prototype to achieve the desired effect

and function calls, sticking closely to the requirements of how the built-in forEach is expected to function.

across all arrays within the scope of execution. **Example Walkthrough**

Let's consider an example where we have an array of names and we want to print each name to the console with a greeting. The

First, we'd set up our custom for Each method by extending the Array prototype as described in the solution approach. This

method would then be available on every array. We would define a callback function that takes currentValue (the name in this context), index, and array as arguments. Our callback function would simply print the greeting to the console using the current name:

Python

Java

class Array(list):

arr = Array([1, 2, 3])

array[index] = value * 2

When we execute this code, our custom for Each method iterates over the array. For each element:

a. It calls the greeting function using call, which applies the given context (if any). In this case, we haven't provided a

b. This results in our greeting function being called with each name and its corresponding index. The greeting function then

We would then call our custom for Each method on our array of names, passing the greeting function as the callback.

- logs the greeting to the console. As the loop progresses, greeting would be called with 'Alice' at index 0, 'Bob' at index 1, and 'Charlie' at index 2.
- Solution Implementation

This small example illustrates how the custom for Each method behaves similarly to the built-in for Each, allowing us to execute a

if context: callback(value, index, self, context) else: # Call the callback with the current element, its index, and the list itself

Define an optional context object (unused in this example) context = {"context": True}

* A utility class to work with arrays

class ArrayUtils {

* @param array

* @param <T>

};

int main() {

};

return 0;

interface Arrav<T> {

// Iterate over each element in the array

callback.call(context, this[i], i, this);

for (let i = 0; i < this.length; i++) {</pre>

const context = { "context": true };

if context:

else:

array[index] = value * 2

context = {"context": True}

arr.forEach(doubleValuesCallback, context);

console.log(arr); // Output will be: [2, 4, 6]

TypeScript

/**

*/

```
* Example usage of the custom forEach function.
*/
public class Main {
   public static void main(String[] args) {
```

C++ #include <iostream> #include <vector> #include <functional> // Declare a template for a custom forEach function template <typename T>

// Define a callback function that doubles the value of each element of the vector

auto doubleValuesCallback = [](int& value, size_t index, std::vector<int>& array){

// In this C++ version, it's not used but provided for compatibility with the original interface

void* context = nullptr; // Apply the custom forEach function to the arr vector using the doubleValuesCallback forEach(arr, doubleValuesCallback, context); // Print the updated vector to the console, which should show doubled values for (const int& value : arr) { std::cout << value << " "; // Output will be: 2 4 6</pre>

// Extending the Array prototype interface to include the custom forEach function

forEach(callback: (value: T, index: number, array: T[]) => void, context?: any): void;

// Custom implementation of the forEach function that adheres to TypeScript's syntax and type safety

Array.prototype.forEach = function<T>(this: T[], callback: (value: T, index: number, array: T[]) => void, context?: any): void {

// Call the callback function with the specified context, passing the current element, its index, and the array itself

// Define an optional context object (not utilized in this example)

// Example usage of the custom forEach function // Create an array of numbers const arr: number[] = [1, 2, 3]; // Define a callback function that doubles the value of each array element const doubleValuesCallback = (value: number, index: number, array: number[]): void => { array[index] = value * 2;

// Define an optional context object (not utilized in this example)

callback(value, index, self, context)

Define a callback function that doubles the value of each list element

def double values callback(value, index, array, context=None):

Define an optional context object (unused in this example)

Update the list element to its doubled value

// Print the updated array to the console, which should show doubled values

class Array(list): # The custom forEach function added to the Array class (inherits from list) def forEach(self, callback, context=None): # Iterate over each element in the list for index, value in enumerate(self): # If context is provided, apply it to the callback

// Apply the custom forEach function to the arr array using the doubleValuesCallback and context

Call the callback with the current element, its index, and the list itself callback(value, index, self) # Example usage of the custom forEach function # Create an instance of Array with numbers arr = Array([1, 2, 3])

Apply the custom forEach function to the arr Array using the doubleValuesCallback and context arr.forEach(double_values_callback, context) # Print the updated list to the console, which should show doubled values print(arr) # Output will be: [2, 4, 6]

Time and Space Complexity

The time complexity of the provided custom for Each function is O(n), where n is the number of elements in the array upon which for Each is called. This is due to the for loop iterating through each element of the array exactly once.