

Problem Description

The problem requires us to extend the functionality of all arrays in TypeScript by adding a custom method called last. This method should return the last element of the array when called. If the array is empty, meaning it has no elements, the method should return -1. It is mentioned that we can assume the array is the result of JSON. parse, which indicates that we are dealing with an array of elements that originated from a JSON string.

The method array.last() will thus become a universal method that can be applied to any array instance within the TypeScript environment after we've added this functionality.

To implement a new method that can be used on all arrays, we can extend the built-in Array prototype in TypeScript. Prototypes in

Intuition

JavaScript (and by extension, TypeScript) are part of the language's prototypal inheritance system. By adding a method to Array. prototype, we actually add that method to all arrays because they inherit methods from their prototype. The last method needs to do the following:

Check if the array is empty. If it is, return –1.

If the array is not empty, return the last element.

return an element of the array's type T or -1.

Here's how the implementation works step-by-step:

We can check if the array is empty by looking at its length property. If length is 0, the array is empty.

To access the last element of the array, we can use the existing method at (-1), which is a part of modern JavaScript that allows us to access the last element by providing -1 as the index. If this at method is not supported or if we need to be compatible with

environments that do not support modern features, we can use this [this.length - 1] to access the last element of the array using the classical approach. By defining Array.prototype.last as a function, we ensure that any array created or existing will have the last method available.

We use TypeScript's declare global to extend the global Array interface so that TypeScript knows about our new method and does not throw type errors when attempting to use last() on an array. Inside this declaration, we indicate that last is either going to

The Array.prototype.last function definition follows this interface, using a ternary operator to either return the last element if the array is not empty or -1 otherwise.

Solution Approach

The solution makes use of the prototypal inheritance feature of JavaScript, as well as TypeScript's ability to declare types and

interfaces that will allow us to extend native objects.

1. Adding a new method to a native object: We start by declaring a global enhancement to the Array type. This is achieved through TypeScript's declare global which will not affect runtime behavior but allows us to tell TypeScript about the new shape

interface Array<T> {

of the Array interface.

1 declare global {

or the number -1.

```
last(): T | -1;
Here we're telling TypeScript that we're adding a new method last that will return either the last element of the array, of type T,
```

1 Array.prototype.last = function () { return this.length ? this.at(-1) : -1; 3 };

3. Logic within the last Method: The last method checks whether the array upon which it's called (this) has any elements by

2. Implementing the last Method: The functionality of this method is attached to the Array prototype. Attaching properties or

methods to a prototype means all instances of the prototype will have access to those properties or methods.

```
checking this length. If the length is not zero, meaning the array has at least one element, it returns the last element using
this.at(-1). Otherwise, if the array is empty (length === 0), it returns -1.
```

4. ESNext Feature: The at method utilized here is a part of ESNext (proposed features for ECMAScript beyond its current edition). It allows for direct access to the array element at a given index, including negative indices which count backwards from the last item.

5. No Reference Solution Approach: The reference solution approach is not provided in the description; however, based on the

problem description and solution code, the aforementioned steps define the algorithm and approach used clearly.

- 6. Exporting as a TypeScript Module: Lastly, the solution includes export {};, which is necessary to treat the file as a module, which in turn allows the global declaration augmentation. Without treating the file as a module, you could unintentionally pollute the global namespace.
- an additional method. The pattern used here is augmentation of a native prototype, which is a powerful feature of JavaScript that needs to be used carefully to avoid unexpected side-effects, especially when extending native objects. **Example Walkthrough**

Algorithmically, this is a very straightforward approach: enhancement involves extending an existing data structure (the Array) with

Suppose we have two arrays, one with several elements, and one that is empty: 1 let numbersArray = [10, 20, 30, 40, 50]; 2 let emptyArray = [];

To utilize the last method that we plan to add to the Array prototype, we would perform the following steps:

instances, including numbersArray and emptyArray.

elements), it will return the last element, which is 50.

declaration does not leak into the global scope unintentionally.

needing to write additional logic each time this common task is required.

1 let lastElementOfNumbers = numbersArray.last(); // expected to return 50

1 let lastElementOfEmpty = emptyArray.last(); // expected to return -1

1. Declare and Define the Last Method: First, we declare the method in the global scope, telling TypeScript about this new method we're going to create. Then, we actually define the method on the Array prototype as shown earlier.

Let's consider a small example to illustrate the solution approach detailed in the content provided.

3. Calling the last Method on our Arrays: • When numbersArray.last() is called, the method checks if numbersArray is empty. Since its length is not zero (it has

2. Applying the last Method to Arrays: After adding the last method to the Array prototype, it becomes available on all array

• When emptyArray.last() is called, the method looks at the length of emptyArray, finds it to be 0, and thus returns -1, indicating that the array is empty.

```
By following these steps, we have enhanced all array instances in our TypeScript environment with a new last method that adheres
to the logic described. This method is now part of the array's prototype chain, so any array created or that exists within the
TypeScript runtime has access to it. The export {} at the end of our TypeScript file ensures that the module's augmented type
```

Using this method, developers can now reliably get the last element of any array or handle the case when the array is empty without

class CustomList(list): def last(self): Returns the last element of the CustomList or -1 if the CustomList is empty.

Check if the list is non-empty, return the last element using negative indexing.

12 # print(array.last()) # Should output 3

11 # array = CustomList([1, 2, 3])

If the list is empty, return -1.

return self[-1] if self else -1

Python Solution

9

13

18

19

20

10 # Usage:

Java Solution

```
import java.util.List;
   public class ArrayUtils {
       // Utility method 'last' that returns the last element of the list or -1 if the list is empty.
       public static <T extends Number> T last(List<T> list) {
           // Check if the list is non-empty, then return the last element; else return -1 according to the list's number type.
           return list.isEmpty() ? (T) Integer.valueOf(-1) : list.get(list.size() - 1);
8
9
10
       // Illustration usage
11
       public static void main(String[] args) {
12
           List<Integer> array = List.of(1, 2, 3);
13
           System.out.println(ArrayUtils.last(array)); // Should output 3
14
15
           List<Integer> emptyArray = List.of();
16
           System.out.println(ArrayUtils.last(emptyArray)); // Should output -1
17
```

// Template class for MyArray which is similar to standard arrays // and provides a 'last' method. 6 template <typename T> class MyArray {

C++ Solution

1 #include <vector>

#include <iostream>

```
private:
       // Use std::vector to handle dynamic arrays.
        std::vector<T> data;
10
   public:
       // Add a new element to the array.
13
       void push(const T& value) {
14
            data.push_back(value);
15
16
17
       // Returns the last element or -1 if the array is empty.
       // The return type is T, assuming T can be -1. For non-numeric
19
       // types, you would have to handle the empty case differently.
20
       T last() const {
21
22
           if (data.empty()) {
                // Assuming T can be -1, if T is numeric.
24
                // This is a simple placeholder for an actual error value
25
                // or behavior to indicate an empty MyArray.
26
                return static_cast<T>(-1);
            } else {
28
                return data.back();
29
30
31
32
       // Other methods to access and modify the array would go here...
33 };
34
   // Usage example for MyArray:
   int main()
       MyArray<int> myArray;
       myArray.push(1);
38
39
       myArray.push(2);
       myArray.push(3);
40
41
        std::cout << "The last element is: " << myArray.last() << std::endl; // Should output 3
42
43
44
        return 0;
```

last(): T | -1;

Typescript Solution

45 }

46

```
1 // Extend the global Array interface with a new 'last' method that returns
 2 // the last element of the array or -1 if the array is empty.
   declare global {
       interface Array<T> {
   // Provide the implementation for the 'last' method defined in the Array interface.
  // It returns the last element of the array or -1 if the array is empty.
11 Array.prototype.last = function () {
       // 'this.length' checks if the array is non-empty,
       // 'this.at(-1)' retrieves the last element,
       // if the array is empty, returns -1.
14
       return this.length > 0 ? this[this.length - 1] : -1;
15
16 };
17
18 // Illustration usage:
  // const array = [1, 2, 3];
  // console.log(array.last()); // Should output 3
21
  // Exporting an empty object to ensure the module system doesn't complain
   // about the lack of exports in this file.
  export {};
Time and Space Complexity
```

Time Complexity

array is not empty, using the Array.prototype.at(-1) method, which is a constant-time operation. **Space Complexity**

The space complexity for the last method is 0(1). The method does not use any additional space that is dependent on the input size. It simply returns the existing value from the array or the number -1.

The time complexity for the last method is 0(1). This is because the method simply accesses the last element of the array if the