// What are Array-like Objects?  
// Introduction  
// JavaScript has "Array-like Objects", which are Object representations of Arrays with a length property. For example:  
  
var *realArray* = ['a', 'b', 'c'];  
*console*.log("Length: ", *realArray*.length);  
var *arrayLike* = {  
 0: 'a',  
 1: 'b',  
 2: 'c',  
 length: 3  
};  
*console*.log("Length: ", *arrayLike*.length);  
  
// Demonstrate that array-like objects are not arrays  
*console*.log("Is array?", *Array*.isArray(*arrayLike*)); // false  
*console*.log("Is array?", *Array*.isArray(*realArray*)); // true  
  
// key features of array-like objects:  
// - They have numeric indices (0, 1, 2, etc.)  
// - They have a length property  
// - They can access elements using bracket notation (obj[0])  
// - They are not iterable by default  
  
// Common Array-like Objects  
// 1.The arguments object  
// 2.The HTMLCollection object  
// 3.The NodeList object  
// Example 1: arguments object in functions  
function exampleArguments() {  
 *console*.log("Arguments length: ", arguments.length);  
 *console*.log("First argument: ", arguments[0]);  
 *console*.log("Arguments object: ", arguments);  
}  
  
exampleArguments('hello', 'world', '!');  
  
// Example 2: HTMLCollection from getElementsByTagName  
// This would work in a browser environment  
if (typeof *document* !== 'undefined') {  
 const divs = *document*.getElementsByTagName('div');  
 *console*.log("HTMLCollection length: ", divs.length);  
 *console*.log("First div: ", divs[0]);  
}  
  
// Example 3: NodeList from querySelectorAll  
// This would work in a browser environment  
if (typeof *document* !== 'undefined') {  
 const paragraphs = *document*.querySelectorAll('p');  
 *console*.log("NodeList length: ", paragraphs.length);  
 *console*.log("First paragraph: ", paragraphs[0]);  
}  
  
/\*  
However, one key difference between Arrays and Array-like Objects is that Array-like objects inherit from  
Object.prototype instead of Array.prototype. This means that Array-like Objects can't access common Array  
prototype methods like forEach(), push(), map(), filter(), and slice():  
 \*/  
// Demonstrate array methods working on a real array  
*console*.log("Real array forEach:");  
*realArray*.forEach(item => *console*.log(item));  
*console*.log("Real array map:", *realArray*.map(x => x.toUpperCase()));  
  
// Try array methods on an array-like object (will fail)  
try {  
 *console*.log("Array-like forEach:");  
 *arrayLike*.forEach(item => *console*.log(item));  
} catch (e) {  
 *console*.log("Error:", e.message);  
}  
  
// Solving the problem  
// Converts array-like to a real array  
// Method 1: Array.from()  
*console*.log("\nConverting array-like to real array using Array.from():");  
const *convertedArray1* = *Array*.from(*arrayLike*);  
*console*.log("Converted array:", *convertedArray1*);  
*console*.log("Is array?", *Array*.isArray(*convertedArray1*)); // true  
*console*.log("Converted array forEach:");  
*convertedArray1*.forEach(item => *console*.log(item)); // Works  
*console*.log("Converted array map:", *convertedArray1*.map(x => x.toUpperCase()));  
  
// Method 2: for loop  
*console*.log("\nConverting array-like to real array using for loop:");  
const *convertedArray2* = [];  
for (let i = 0; i < *arrayLike*.length; i++) {  
 *convertedArray2*.push(*arrayLike*[i]);  
}  
*console*.log("Converted array:", *convertedArray2*);  
*console*.log("Is array?", *Array*.isArray(*convertedArray2*)); // true  
*convertedArray2*.forEach(item => *console*.log(item)); // Works  
*console*.log("Converted array map:", *convertedArray2*.map(x => x.toUpperCase())); // Works  
  
// Method 3: Object.keys  
*console*.log("\nConverting array-like to real array using Object.keys:");  
const *convertedArray3* = *Object*.keys(*arrayLike*).map(key => *arrayLike*[key]);  
*console*.log("Converted array:", *convertedArray3*);  
*console*.log("Is array?", *Array*.isArray(*convertedArray3*)); // true  
*convertedArray3*.forEach(item => *console*.log(item)); // Works  
  
// Note: This method only works if the array-like object has a numerically indexed property for each index.  
// Otherwise, it will return all enumerable property names, including both numeric indices AND the "length" property (and any other properties you might have added).  
// Therefore, when mapping over these keys, you'll get all values, including the "length" value, which you might not want.  
// For that reason, using the map() method on the converted array was impossible to modify its elements (converting to uppercase like in previous methods).  
  
// Method 4: Use Array.prototype.slice  
*console*.log("\nConverting array-like to real array using Array.prototype.slice:");  
const *convertedArray4* = *Array*.prototype.slice.call(*arrayLike*);  
*console*.log("Converted array: ", *convertedArray4*);  
*console*.log("Is array? ", *Array*.isArray(*convertedArray4*)); // true  
*convertedArray4*.forEach(item => *console*.log(item)); // Works  
*console*.log("Converted array map:", *convertedArray4*.map(x => x.toUpperCase())); // Works  
  
// Conclusion  
// Array-like objects are objects that have a length property, but are not arrays.  
// They are useful for representing arrays in environments where arrays are not available (e.g. Node.js).  
// However, they are not arrays, and they do not have built-in array methods.  
// To convert them to arrays, you can use the Array.from() method, use a for loop, Object.keys, or Array.prototype.slice.  
// Note that the spread operator could not be used to convert an array-like object to an array because they are not iterable by default.