## 1. Advanced Coding: Implement `customFilter`

Create a function `customFilter` that mimics the behavior of the JavaScript `Array.prototype.filter` method. It should not use the built-in `filter` method. The function should handle cases where the callback function returns truthy or falsy values, not just `true` or `false`.

## 2. Complex Debugging: Analyze Recursive Reduce

The following recursive `reduce` function is intended to flatten an array of arrays into a single array, but it contains errors. Identify and fix them:

```
function flattenArray(arr) {
    return arr.reduce((acc, val) => {
        if (Array.isArray(val)) {
            acc.concat(flattenArray(val));
        } else {
            acc.push(val);
        }
        return acc;
    }, []);
}
```

## 3. Predict Output: Chaining Array Methods

What is the output of the following code snippet? Explain the process.

```
const data = [1, '2', 3, '4', 5];
const result = data
    .map(item => parseInt(item))
    .filter(item => !isNaN(item))
    .reduce((acc, item) => acc + item, 0);
console.log(result);
```

## 4. Advanced Coding: Nested Object Transformation

Write a function that takes an array of objects with nested objects and transforms each nested object's key-value pairs into arrays of `[key, value]`. Use `map` and `reduce`.

## 5. Complex Predict Output: forEach with Objects

Given the following code, what will be logged to the console?

```
const obj = { a: 1, b: 2, c: 3 };
Object.keys(obj).forEach(key => {
    obj[key] *= 2;
});
console.log(obj);
```

## 6. Advanced Debugging: Complex Array.from Usage

The following code is intended to create an array of objects from an array-like object, but it doesn't work as expected. Identify and fix the issues:

```
const arrayLike = {0: 'apple', 1: 'banana', 2: 'cherry', length: 3};
const fruits = Array.from(arrayLike, key => ({name: key}));
console.log(fruits);
```

## 7. Coding Challenge: Multi-level Grouping

Using `reduce`, write a function that takes an array of objects and groups them by a specified key, and within each group, further groups them by another key. The function should be flexible to work with any keys.

## 8. Predict Output: Interplay of Map, Filter, and Reduce

Analyze and determine the output of the following complex sequence of `map`, `filter`, and `reduce`:

```
const numbers = [1, 2, 3, 4, 5, 6];
const result = numbers
    .map(num => num * 2)
    .filter(num => num % 3 === 0)
    .reduce((acc, num) => acc + num, 0);
console.log(result);
```

## 9. Advanced Coding: Array-Like Object Operations

Create a function that takes an array-like object and a callback function. This function should behave like the `forEach` method, applying the callback to each property of the object that represents an array element.

## 10. Complex Debugging: Reduce with Initial Object

The following code is supposed to create an object that counts the occurrences of each string in an array, but it has errors. Fix the code:

```
const strings = ['apple', 'banana', 'apple', 'orange', 'banana', 'apple'];
const count = strings.reduce((acc, str) => {
    acc[str] = acc[str] ? acc[str]++ : 1;
    return acc;
}, {});
console.log(count);
```