Q1. Explain the .map() function in JavaScript and provide three examples with detailed explanations.

The .map() function in JavaScript is a powerful array method that allows you to transform each element of an array into a new element, returning a new array with the same length as the original one. It applies a given function to each element in the array, creating a new array with the results of calling that function on every element.

Example 1: Converting an Array of Numbers from Celsius to Fahrenheit

```
const celsiusTemps = [0, 10, 20, 30, 40];

const fahrenheitTemps = celsiusTemps.map(function(temp) {
  return temp * 9/5 + 32;
});

console.log(fahrenheitTemps); // Output: [32, 50, 68, 86, 104]
```

Example 2: Doubling the Numbers in an Array

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

const doubledNumbers = numbers.map(function(num) {
  return num * 2;
});

console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]
```

Example 3: Extracting Specific Properties from an Array of Objects

```
const people = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 },
  { name: 'Charlie', age: 35 }
];

const names = people.map(function(person) {
  return person.name;
});

console.log(names); // Output: ['Alice', 'Bob', 'Charlie']
```

Key Points

- .map() does not modify the original array; it returns a new array with the transformed elements.
- It is ideal for cases where you want to apply the same operation to every element in an array and get a new array with the results.
- The length of the new array is the same as the original array.

Q2. Explain the .reduce() function in JavaScript and provide three examples with detailed explanations.

The .reduce() function in JavaScript is a powerful array method that allows you to reduce an array to a single value by applying a function to each element in the array (from left to right). The function accumulates a result by processing each element and returns the final accumulated value.

Example 1: Summing All Elements in an Array

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

const sum = numbers.reduce(function(accumulator, currentValue) {
   return accumulator + currentValue;
}, 0);


console.log(sum); // Output: 15
```

Example 2: Finding the Maximum Value in an Array

```
const numbers = [10, 5, 20, 15];

const max = numbers.reduce(function(accumulator, currentValue) {
   return currentValue > accumulator ? currentValue : accumulator;
}, numbers[0]);


console.log(max); // Output: 20
```

Example 3: Counting Occurrences of Values in an Array

```
const fruits = ['apple', 'banana', 'orange', 'apple', 'orange', 'banana', 'banana'];

const fruitCount = fruits.reduce(function(accumulator, currentValue) {
   if (accumulator[currentValue]) {
      accumulator[currentValue]++;
   } else {
      accumulator[currentValue] = 1;
   }
   return accumulator;
}, {});


console.log(fruitCount); // Output: { apple: 2, banana: 3, orange: 2 }
```

Key Points

- .reduce() is ideal for scenarios where you need to compute a single value from an array, such as summing elements, finding the maximum, or accumulating data into an object.
- The initialValue is optional but important; without it, the first element of the array is used as the initial accumulator, which can lead to unexpected results if the array is empty or if you're dealing with non-numeric data.
- The return value of the .reduce() function is the final accumulated value after processing all elements in the array.

Q3. Explain the .filter() function in JavaScript and provide three examples with detailed explanations.

The .filter() function in JavaScript is an array method that creates a new array containing all the elements of the original array that pass a specified test (provided as a function). It is often used when you need to extract a subset of elements from an array that meet certain conditions.

Example 1: Filtering Even Numbers from an Array

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

const evenNumbers = numbers.filter(function(num) {
   return num % 2 === 0;
});

console.log(evenNumbers); // Output: [2, 4, 6, 8, 10]
```

Example 2: Filtering Words Longer Than 5 Characters

```
const words = ['apple', 'banana', 'cherry', 'date', 'elderberry'];

const longWords = words.filter(function(word) {
  return word.length > 5;
});

console.log(longWords); // Output: ['banana', 'cherry', 'elderberry']
```

Example 3: Filtering Objects Based on a Property

```
const people = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 },
  { name: 'Charlie', age: 35 },
  { name: 'David', age: 28 }
];

const adults = people.filter(function(person) {
  return person.age >= 30;
});

console.log(adults); // Output: [ { name: 'Bob', age: 30 }, { name: 'Charlie', age: 35 } ]
```

Key Points

- The .filter() method does not modify the original array; instead, it returns a new array with elements that pass the provided test.
- If no elements pass the test, .filter() returns an empty array.
- It's a useful method when you need to create a subset of elements from an array based on specific conditions.