JavaScript Coding Exercises

JavaScript coding exercise that involves working with arrays and string manipulation:

Exercise: Write a function that takes in an array of words and returns a new array containing only the words that have all their letters in alphabetical order.

Example:

```
const words = ['hello', 'world', 'abcedf', 'goodbye'];
console.log(wordsInOrder(words)); // ['abcedf']
```

Explanation:

To solve this problem, we can use the filter() method to iterate through the original array and return a new array containing only the words that meet the criteria. We can check if a word has all its letters in alphabetical order by splitting the word into an array of characters, sorting that array, and then comparing it to the original array. If the two arrays are equal, then all the letters in the word are in alphabetical order.

Here's the code for the solution:

```
function wordsInOrder(arr) {
  return arr.filter(word => {
    const sortedWord = word.split('').sort().join('');
    return sortedWord === word;
  });
}
```

In the above code, the split() method splits the word into an array of characters, the sort() method sorts that array, and the join() method joins the sorted array back into a string. We then compare the sorted string to the original word to determine if all the letters are in alphabetical order. If they are, the filter() method includes the word in the new array that is returned.

We can test this function with the example provided earlier:

```
const words = ['hello', 'world', 'abcedf', 'goodbye'];
console.log(wordsInOrder(words)); // ['abcedf']
```

This will output ['abcedf'] because it's the only word in the original array that has all its letters in alphabetical order.

JavaScript coding exercise that involves working with arrays and conditional statements:

Exercise: Write a function that takes in an array of numbers and returns the second highest number in the array.

Example:

```
const nums = [10, 2, 5, 8, 9];
console.log(findSecondHighest(nums)); // 9
```

Explanation:

To solve this problem, we can first sort the array in descending order using the sort() method. Once sorted, we can then return the second element in the array. However, we must first check that the array has at least two elements to prevent errors. If the array has less than two elements, we'll return an error message.

Here's the code for the solution:

```
function findSecondHighest(arr) {
  if (arr.length < 2) {
    return 'Error: array must have at least 2
elements';
  }

  const sortedArr = arr.sort((a, b) => b - a);
  return sortedArr[1];
}
```

In the above code, we first check if the array has less than two elements. If it does, we return an error message. If the array has two or more elements, we sort it in descending order using the sort() method and a custom compare function (a, b) => b - a that sorts the array in descending order. We then return the second element in the sorted array using sortedArr[1].

We can test this function with the example provided earlier:

```
const nums = [10, 2, 5, 8, 9];
console.log(findSecondHighest(nums)); // 9
```

This will output 9 because it's the second highest number in the original array.

JavaScript coding exercise that involves working with objects and loops:

Exercise: Write a function that takes in an array of objects representing people and returns an array of their full names in alphabetical order.

Each object in the array will have the following properties:
firstName (string): the person's first name

lastName (string): the person's last name

Example:

```
const people = [
  { firstName: 'John', lastName: 'Doe' },
  { firstName: 'Jane', lastName: 'Doe' },
  { firstName: 'Alice', lastName: 'Smith' }
];
console.log(alphabeticalNames(people)); // ['Alice
Smith', 'Jane Doe', 'John Doe']
```

Explanation:

To solve this problem, we can first use the map() method to iterate over the array of people and create a new array of their full names. We can then sort this new array using the sort() method with a custom compare function. Finally, we can return the sorted array.

Here's the code for the solution:

```
function alphabeticalNames(arr) {
  const fullNames = arr.map(person =>
`${person.firstName} ${person.lastName}`);
  fullNames.sort((a, b) => a.localeCompare(b));
  return fullNames;
}
```

In the above code, the map() method iterates over the array of people and creates a new array of their full names by concatenating the firstName and lastName properties with a space in between. The sort() method then sorts this new array in alphabetical order using the localeCompare() method. Finally, the sorted array is returned.

We can test this function with the example provided earlier:

```
const people = [
  { firstName: 'John', lastName: 'Doe' },
  { firstName: 'Jane', lastName: 'Doe' },
  { firstName: 'Alice', lastName: 'Smith' }
];

console.log(alphabeticalNames(people)); // ['Alice Smith', 'Jane Doe', 'John Doe']
```

This will output ['Alice Smith', 'Jane Doe', 'John Doe'] because the array of full names has been sorted in alphabetical order.

JavaScript coding exercise that involves working with strings and loops:

Exercise: Write a function that takes in a string and returns a new string with the first letter of each word capitalized.

Example:

```
console.log(capitalizeWords('hello world')); // 'Hello World'
console.log(capitalizeWords('the quick brown fox')); // 'The Quick Brown Fox'
```

Explanation:
To solve this problem, we can split the input string into an array of words using the split() method. We can then use a loop to iterate over each word in the array, capitalize the first letter of each word using the toUpperCase() method, and concatenate the rest of the word using the slice() method. Finally, we can join the modified array back into a string using the join() method and return it.

Here's the code for the solution:

```
function capitalizeWords(str) {
  const words = str.split(' ');
```

```
  for (let i = 0; i < words.length; i++) {
    words[i] = words[i][0].toUpperCase() +
words[i].slice(1);
  }
  return words.join(' ');
}
```

In the above code, we first split the input string into an array of words using the split() method with a space delimiter. We then use a for loop to iterate over each word in the array. For each word, we capitalize the first letter using the toUpperCase() method and concatenate the rest of the word using the slice() method. The modified word is then stored back into the array at the same index using words[i] = ...

Finally, we join the modified array back into a string using the join() method with a space delimiter and return it.

We can test this function with the examples provided earlier:

```
console.log(capitalizeWords('hello world')); // 'Hello
World'
console.log(capitalizeWords('the quick brown fox')); //
'The Quick Brown Fox'
```

These will output 'Hello World' and 'The Quick Brown Fox' respectively, since the function correctly capitalizes the first letter of each word in the input string.

JavaScript coding exercise that involves working with arrays and conditional statements:

Exercise: Write a function that takes in an array of numbers and returns the largest difference between any two elements where the larger element comes after the smaller element.

Example:

```
console.log(largestDifference([4, 3, 10, 2, 8])); // 8
console.log(largestDifference([7, 8, 2, 5, 10, 6])); // 8
```

Explanation:

To solve this problem, we can use a for loop to iterate over the array of numbers and keep track of the smallest number seen so far (min) and the largest difference seen so far (maxDiff). For each number in the array, we can calculate the difference

between that number and min and update maxDiff if it's larger than the current value.

Here's the code for the solution:

```
function largestDifference(arr) {
  let min = arr[0];
  let maxDiff = 0;

  for (let i = 1; i < arr.length; i++) {
    if (arr[i] < min) {
      min = arr[i];
    } else {
      maxDiff = Math.max(maxDiff, arr[i] - min);
    }
  }

  return maxDiff;
}
```

In the above code, we first initialize min to the first element of the array and maxDiff to 0. We then use a for loop to iterate over the array starting at index 1. For each element in the array, we check if it's smaller than min. If it is, we update min to that element. If it's larger than min, we calculate the difference between the element and min using subtraction, and update maxDiff if the difference is larger than the current value.

Finally, we return maxDiff, which will contain the largest difference between any two elements where the larger element comes after the smaller element.

We can test this function with the examples provided earlier:

```
console.log(largestDifference([4, 3, 10, 2, 8])); // 8
console.log(largestDifference([7, 8, 2, 5, 10, 6])); // 8
```

These will output 8 for both examples, since the largest difference between any two elements where the larger element comes after the smaller element is 8 in both cases.