Coding Challenges

Assignment Questions

Assignment Answer

1. Count the occurrences. You are building a word count generator that will take a large string of text as input and output the words and the number of times they are present in the string. Your task is to write a function that can count the occurrences of each word in the string and return a Map containing each word's count. For simplicity, you can assume that the string only contains alphabetic characters and spaces.

Ans:- To achieve this in JavaScript, you can use the `setTimeout()` function to create a delay before reversing the string. Below is a simple implementation:

Javascript

Code

```
// Define the input string

let input = "Hello, World!";



// Function to reverse the string

function reverseString(str) {

    return str.split("").reverse().join("");

}

// Use setTimeout to reverse the string after a
delay of 2 seconds (2000 milliseconds)

setTimeout(function() {

    let reversedString = reverseString(input);
```

```
    console.log(reversedString);

}, 2000);
```

```
OUTPUT- !dlroW ,olleH
```

Explanation:

1. **Input String: The variable `input` stores the string to be reversed.**
2. **reverseString Function: This function takes a string, splits it into an array of characters, reverses the array, and then joins it back into a string.**
3. **setTimeout Function: This function is used to execute the reversal after a delay of 2 seconds (2000 milliseconds). The reversed string is then logged to the console.**

When you run this code, you'll see the reversed string printed to the console after a 2-second delay.

2. Only unique items are allowed. You are building a program that takes an array of numbers as input and you need to remove all the duplicates from the array. You want to write a function that can accomplish this task efficiently and returns a new set that contains only the unique elements.

Ans:- Here's a JavaScript program that generates a random number after a delay of 3 seconds. It also shows a countdown each second indicating the time remaining until the random number is generated:

javascript

Code

```
// Define the delay in seconds

let delayInSeconds = 3;

// Function to generate a random number

function generateRandomNumber() {

    return Math.floor(Math.random() * 100) + 1;
// Generates a random number between 1 and 100

}
```

```
// Function to display the countdown and generate the random number

function startCountdownAndGenerateNumber() {

    let remainingTime = delayInSeconds;



    // Set an interval to display the countdown every second

    let countdownInterval = setInterval(function() {

        console.log(`Time remaining: ${remainingTime} seconds`);

        remainingTime--;



        // When the countdown reaches 0, clear the interval and generate the random number
```

```
        if (remainingTime < 0) {

            clearInterval(countdownInterval);

            let randomNumber =
generateRandomNumber();

            console.log(`Generated random number:
${randomNumber}`);

        }

    }, 1000); // Interval is set to 1000
milliseconds (1 second)

}

// Start the countdown and random number
generation

startCountdownAndGenerateNumber();
```

Explanation:

1. **Delay in Seconds: The variable `delayInSeconds` is set to 3 seconds. You can modify this value to change**

the delay.

2. generateRandomNumber Function: This function generates a random number between 1 and 100.
3. startCountdownAndGenerateNumber Function:
    - This function starts a countdown using `setInterval()`, which prints the remaining time every second.
    - When the countdown reaches 0, the interval is cleared using `clearInterval()`, and the random number is generated.
    - The generated random number is then printed to the console.
4. startCountdownAndGenerateNumber(): This function call starts the process.

Expected Output:

When you run this code, the console will show something like this:

Code

```
Time remaining: 3 seconds

Time remaining: 2 seconds

Time remaining: 1 second
```

```
Time remaining: 0 seconds

Generated random number: 42
```

You can copy and paste the code into your browser's console or an online JavaScript editor to see it in action.

3. Swap the values. You are working on a project that requires you to swap the values of two variables without using a temporary variable. You decide to write a function that takes two variables as input and swaps their values using destructuring assignment with an array. The function should take the two variables as arguments, destructure them into an array, and then swap their positions within the array. Finally, the function should return an array with the swapped values. For example, if the function is called with the arguments x = 5 and y = 10, it should return an array [10, 5] with x now equal to 10 and y equal to 5.

Ans:- To create a feature that converts the prices of items from USD to INR using the exchange rate of 1 USD = 80 INR, you can use JavaScript's `map` function. However, since `map` is typically used with arrays, we'll need to use `Object.entries()` to transform the

object into an array, perform the conversion, and then convert it back into an object.

Here's how you can implement this:

javascript

Code

```
// Original store inventory with prices in USD

const storeInventory = {

    "apple": 1.2,

    "banana": 0.5,

    "orange": 0.75,

    "grapes": 2.5,

    "mango": 1.5

};

// Exchange rate from USD to INR

const exchangeRate = 80;
```

```
// Function to convert prices to INR

function convertPricesToINR(inventory, rate) {

    return Object.fromEntries(

        Object.entries(inventory).map(([item,
price]) => {

            return [item, price * rate];

        })

    );

}

// Convert the store inventory prices to INR

const inventoryInINR =
convertPricesToINR(storeInventory, exchangeRate);


// Output the converted inventory

console.log(inventoryInINR);
```

Explanation:

1. **storeInventory: This is the original object where the keys are item names, and the values are prices in USD.**
2. **exchangeRate: The conversion rate from USD to INR.**
3. **convertPricesToINR Function:**
   - **`Object.entries(inventory)`: Converts the `storeInventory` object into an array of key-value pairs.**
   - **`map(([item, price]) => {...})`: Iterates over each key-value pair and multiplies the price by the exchange rate.**
   - **`Object.fromEntries(...)`: Converts the array of key-value pairs back into an object.**
4. **inventoryInINR: The new object with prices converted to INR.**

Output:

When you run the above code, the `inventoryInINR` object will look like this:

javascript

Code

```
{
    apple: 96,
    banana: 40,
    orange: 60,
    grapes: 200,
    mango: 120
}
```

This will be printed to the console as:

javascript

Code

```
{
  apple: 96,
  banana: 40,
  orange: 60,
  grapes: 200,
```

```
  mango: 120
}
```

Each item's price has been converted from USD to INR using the provided exchange rate.

4. Access random elements. You are building a program that takes an array of numbers as input and you need to extract the first, second, and last elements of the array. You decide to write a function that can accomplish this task using destructuring assignment with an array. The function should take the array as an argument, destructure it into three variables representing the first, second, and last elements, and then return an array with these three values. For example, if the input array is [1, 2, 3, 4, 5], the function should return an array [1, 2, 5]

Ans:- You can achieve this by using destructuring assignment in JavaScript. Here's how you can write the function to extract the first, second, and last elements of an array:

javascript

Code

```
function extractElements(arr) {
```

```
    // Destructure the first and second elements
    const [first, second, ...rest] = arr;
    // Get the last element using the length of the array
    const last = rest.pop();


    // Return an array containing the first, second, and last elements
    return [first, second, last];
}
// Example usage
const inputArray = [1, 2, 3, 4, 5];
const result = extractElements(inputArray);
console.log(result);  // Output: [1, 2, 5]
```

Explanation:

1. **Destructuring Assignment:**
    - `const [first, second, ...rest] = arr;`:
        - This line destructures the array into `first`, `second`, and `rest`.
        - `first` is the first element of the array.
        - `second` is the second element of the array.
        - `...rest` is an array containing all the remaining elements after the first two.
2. **Accessing the Last Element:**
    - `const last = rest.pop();`:
        - This line extracts the last element from the `rest` array by using the `pop()` method, which removes and returns the last element of an array.
3. **Returning the Result:**
    - The function returns a new array `[first, second, last]`, which contains the first, second, and last elements.

Example:

If the input array is `[1, 2, 3, 4, 5]`, the output will be:

javascript

Code

```
[1, 2, 5]
```

> **This approach ensures that you can handle arrays of any length, and it correctly returns the first, second, and last elements of the input array.**

.

5. Min and max values. You are building a program that needs to find the maximum and minimum values in an array of numbers. You decide to write a function that can accomplish this task by taking an array of numbers as input and using the spread operator with the Math methods max() and min(). The function should return an object with two properties, max, and min, representing the maximum and minimum values in the input array, respectively. For example, if the input array is [5, 2, 7, 1, 9], the function should return an object { max: 9, min: 1 }.

Ans:- You can achieve this by using the spread operator with the `Math.max()` and `Math.min()` methods in JavaScript. Here's how

you can write a function to find the maximum and minimum values in an array:

javascript

Code

```
function findMinMax(arr) {

    // Use the spread operator with Math.max and Math.min to find the max and min values

    const max = Math.max(...arr);

    const min = Math.min(...arr);


   // Return an object with properties max and min

    return { max, min };

}

// Example usage

const inputArray = [5, 2, 7, 1, 9];
```

```
const result = findMinMax(inputArray);

console.log(result);  // Output: { max: 9, min: 1
}
```

Explanation:

1. **Spread Operator:**
    - **The spread operator (`...`) is used to spread out the elements of the array as individual arguments to the `Math.max()` and `Math.min()` functions.**
    - **`Math.max(...arr)` finds the maximum value in the array.**
    - **`Math.min(...arr)` finds the minimum value in the array.**
2. **Return an Object:**
    - **The function returns an object with two properties:**
        - **`max`: The maximum value found in the array.**
        - **`min`: The minimum value found in the array.**

Example:

If the input array is `[5, 2, 7, 1, 9]`, the output will

be:

Javascript

Code

```
{ max: 9, min: 1 }
```

This function is efficient and concise, making use of the spread operator to easily find the maximum and minimum values in the array.

6. Nested Objects. Write a function that takes an object representing a person as input, and extracts the name and street properties from a nested object using object destructuring. The function should return an object with these two properties. A sample object is given below.

Ans:- To extract the `name` and `street` properties from a nested object using object destructuring, you can write a function that does the following:

Sample Object:

javascript

Code

```
const person = {
    name: "John Doe",
    age: 30,
    address: {
        street: "123 Main St",
        city: "Anytown",
        zip: "12345"
    }
};
```

Function Implementation:

javascript

Code

```
function extractNameAndStreet(person) {
    // Destructure the name from the person
```

```
object and street from the nested address object

    const { name, address: { street } } = person;


   // Return an object with name and street
properties

    return { name, street };

}

// Example usage

const extractedInfo =
extractNameAndStreet(person);

console.log(extractedInfo);  // Output: { name:
'John Doe', street: '123 Main St' }
```

Explanation:

1. **Destructuring:**
    - `const { name, address: { street } } = person;`:
        - `name` **is directly destructured from the**

`person` object.

- **`street` is destructured from the nested `address` object inside the `person` object.**

2. Returning the Result:

- **The function returns an object containing the `name` and `street` properties.**

Example Output:

When you run the function with the sample object, the output will be:

javascript

Code

```
{ name: 'John Doe', street: '123 Main St' }
```

This approach cleanly and efficiently extracts the required properties from a nested object.