Web Design and Development Guide

Common HTML Mistakes 1

Missing DOCTYPE declaration:

Solution: Add a DOCTYPE declaration to the top of your HTML document to specify the version of HTML you are using.

Unclosed tags:

Solution: Check that all tags have been closed correctly, using the appropriate opening and closing tags.

Improper nesting of tags:

Solution: Make sure that all tags are properly nested within each other, so that there are no overlapping or mismatched tags.

Missing alt text for images:

Solution: Add an alt attribute to all image tags, describing the image for users who cannot see it.

Using deprecated tags or attributes:

Solution: Replace deprecated tags or attributes with newer, more appropriate HTML elements.

Using inline styles instead of CSS:

Solution: Move inline styles into a separate CSS stylesheet, to make your HTML code cleaner and more maintainable.

Incorrectly formatted links:

Solution: Check that all links are formatted correctly, with the correct href attribute and a descriptive link text.

Using non-semantic markup:

Solution: Use semantic HTML elements to describe the content of your page, making it easier for search engines and assistive technology to understand.

Using tables for layout:

Solution: Use CSS for layout, instead of tables, to create more flexible and responsive designs.

Ignoring accessibility:

Solution: Follow accessibility best practices, including adding alt text to images, providing captions for videos, and making sure your site is keyboard-friendly.

Common HTML Mistakes 2

Using non-ASCII characters:

Solution: Use ASCII characters to ensure maximum compatibility across different devices and browsers.

Using inline JavaScript:

Solution: Move JavaScript code into a separate file or use a modern framework like React or Vue to create more organized and maintainable code.

Using duplicate IDs:

Solution: Use unique IDs for each element on a page to prevent conflicts and ensure that JavaScript and CSS selectors work as intended.

Not specifying character encoding:

Solution: Specify the character encoding of your HTML document using the meta charset tag to ensure that special characters are displayed correctly.

Overusing heading tags:

Solution: Use heading tags (h1-h6) appropriately and make sure each one is only used once per page to avoid confusion for users and search engines.

Ignoring responsive design:

Solution: Use responsive design techniques such as media queries to make sure your site looks good on all devices and screen sizes.

Forgetting to close self-closing tags:

Solution: Make sure that all self-closing tags are closed with a forward slash (/) at the end, such as .

Using too many nested elements:

Solution: Avoid using too many nested elements as it can slow down your page load time and make your code more difficult to read and maintain.

Using non-standard HTML:

Solution: Stick to standard HTML syntax and avoid using non-standard elements or attributes that may not be recognized by all browsers.

Not optimizing images:

Solution: Optimize images by reducing their size and using appropriate file formats to improve page load times and overall user experience.

Common HTML Mistakes 3

Using outdated code:

Solution: Keep up-to-date with HTML standards and avoid using outdated or deprecated code.

Ignoring SEO best practices:

Solution: Use appropriate title tags, meta descriptions, and header tags to optimize your website for search engines.

Overusing div tags:

Solution: Use semantic HTML elements instead of overusing div tags to make your code more organized and easier to read.

Using too many inline styles:

Solution: Use external stylesheets or internal style tags instead of using too many inline styles to make your code more maintainable.

Not using the correct file extensions:

Solution: Make sure that you use the correct file extensions for HTML files (e.g., .html, .htm) to ensure that your pages display correctly.

Using too many different fonts:

Solution: Use a limited number of fonts to improve readability and consistency throughout your website.

Not using proper indentation:

Solution: Use proper indentation and whitespace to make your code easier to read and understand.

Ignoring accessibility requirements:

Solution: Make sure that your website meets accessibility requirements, such as providing alt text for images and using ARIA roles and attributes.

Using invalid HTML:

Solution: Validate your HTML using online tools to ensure that it is free of errors and conforms to HTML standards.

Ignoring cross-browser compatibility:

Solution: Test your website on multiple browsers and devices to ensure that it displays correctly for all users.

Web Page Examples CSS HTML

Div 1	Div 2
Div 3	Div 4

To create a CSS grid for four child divs that are arranged in a two-by-two grid within a parent element, you can use the following code:

HTML:

```
<div class="grid-container">
  <div class="grid-item">Div 1</div>
  <div class="grid-item">Div 2</div>
  <div class="grid-item">Div 3</div>
  <div class="grid-item">Div 4</div>
</div>
```

CSS:

```
.grid-container {
  display: grid;
  grid-template-columns: repeat(2, 1fr);
  grid-template-rows: repeat(2, 1fr);
  gap: 10px;
}

.grid-item {
  background-color: #f1f1f1;
  padding: 20px;
  text-align: center;
}
```

In this code, we first create a parent element with the class grid-container. We set its display property to grid, which makes it a grid container. We then set the grid-template-columns property to repeat(2, 1fr) and grid-template-rows to repeat(2, 1fr), which creates a grid with two rows and two columns. Finally, we set the gap property to 10px to create a gap between the grid items.

We then create the four child divs with the class grid-item, which we want to arrange in the grid. We set their background color to #f1f1f1 and add some padding to create some space between the content and the edges of the div. Finally, we center the text within the div using the text-align property.

With Responsive Navigation menu

Regular page over 600px wide

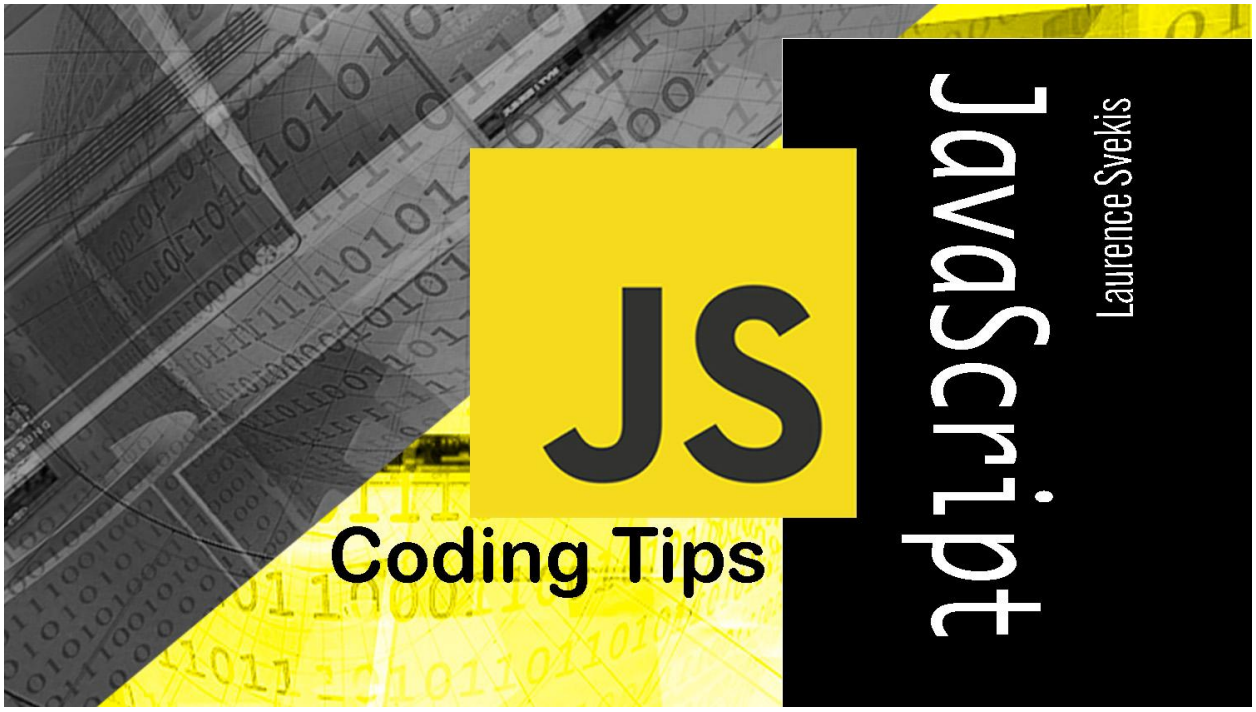
```html
<!DOCTYPE html>
<html>
<head>
  <title>Nav Menu</title>
  <style>
    .navbar {
      background-color: #333;
      overflow: hidden;
    }

    .navbar a {
      color: #f2f2f2;
      text-decoration: none;
      font-size: 1.5em;
      padding: 14px 16px;
      float: left;
      display: block;
      text-align: center;
    }

    .navbar a:hover {
      background-color: #ddd;
      color: black;
    }

    .navbar .icon {
      display: none;
    }
```

```
@media screen and (max-width: 600px) {
    .navbar a:not(:first-child) {
        display: none;
    }

    .navbar .icon {
        display: block;
        float: right;
    }

    .navbar.responsive {
        position: relative;
    }

    .navbar.responsive .icon {
        position: absolute;
        right: 0;
        top: 0;
    }

    .navbar.responsive a {
        float: none;
        display: block;
        text-align: left;
    }
}

.grid-container {
    display: grid;
```

```
        grid-template-columns: repeat(2, 1fr);
        grid-template-rows: repeat(2, 1fr);
        gap: 10px;
    }

    .grid-item {
        background-color: #f1f1f1;
        padding: 20px;
        text-align: center;
    }
  </style>
</head>

<body>
  <nav class="navbar">
    <a href="#" class="icon">&#9776;</a>
    <a href="#">Home</a>
    <a href="#">About</a>
    <a href="#">Services</a>
    <a href="#">Portfolio</a>
    <a href="#">Blog</a>
    <a href="#">Contact</a>

  </nav>


  <div class="grid-container">
    <div class="grid-item">Div 1</div>
    <div class="grid-item">Div 2</div>
```

```
        <div class="grid-item">Div 3</div>
        <div class="grid-item">Div 4</div>
    </div>
    <script>
        document.querySelector('.icon').onclick = function () {
            const ele = document.querySelector('.navbar');
            if (ele.className === 'navbar') {
                ele.className += ' responsive';
            }
        }
    </script>
</body>
</html>
```

JavaScript Best Practices



JavaScript best practices refer to a set of guidelines and recommendations that developers should follow while writing JavaScript code. These best practices are intended to ensure that code is efficient, maintainable, and reliable. Here are some examples of JavaScript best practices with explanations:

Use Strict Mode:

Use the "use strict" directive to enable strict mode in your JavaScript code. This mode helps prevent common coding mistakes and makes it easier to write more secure and maintainable code. For example:

```
"use strict";
// Your JavaScript code here
```

Declare Variables Properly:

Always use the "let" or "const" keywords to declare variables. Avoid using the "var" keyword because it has some confusing scoping rules. For example:

```
let myVar = "Hello World";
const myConst = 42;
```

Use Descriptive Names:

Use descriptive names for your variables and functions. This makes your code more readable and easier to understand. For example:

```
// Bad example:
let x = 5;

// Good example:
let numberOfItems = 5;
```

Use Comments:

Use comments to explain what your code is doing. This makes it easier for other developers to understand your code and helps you remember what you were thinking when you wrote the code. For example:

```
// Calculate the area of a rectangle
let width = 5;
let height = 10;
let area = width * height; // The result is 50
```

Use Templates Instead of Concatenation:

Use template literals instead of concatenation when building strings. This makes your code more readable and easier to maintain. For example:

```
// Bad example:
let name = "John";
let message = "Hello, " + name + "!";

// Good example:
let name = "John";
let message = `Hello, ${name}!`;
```

Use Strict Comparison:

Use the strict equality operator ("===") instead of the loose equality operator ("=="). The strict equality operator compares both the value and data type of two variables. This avoids unexpected results due to type coercion. For example:

```
// Bad example:
if (5 == "5") {
  // This code will execute because "5" is coerced to a
number
}

// Good example:
```

```
if (5 === "5") {
  // This code will not execute because the types are
different
}
```

Avoid Global Variables:

Avoid using global variables because they can cause naming conflicts and make your code harder to maintain. Instead, use local variables and pass them as parameters to functions. For example:

```
// Bad example:
let myVar = 5;

function myFunction() {
  console.log(myVar);
}

// Good example:
function myFunction(myVar) {
  console.log(myVar);
}

myFunction(5);
```

Use Object and Array Destructuring:

Use object and array destructuring to make your code more concise and readable. For example:

```
// Bad example:
let user = { name: "John", age: 30 };
let name = user.name;
let age = user.age;

// Good example:
let user = { name: "John", age: 30 };
let { name, age } = user;
```

Use Arrow Functions:

Use arrow functions to make your code more concise and readable. Arrow functions have a simpler syntax and automatically bind the "this" keyword to the parent scope. For example:

```
// Bad example:
function myFunction() {
  return "Hello World";
}

// Good example:
let myFunction = () => "Hello World";
```

Use Promises and Async/Await:

Use promises and async/await to handle asynchronous operations in JavaScript. Promises allow you to handle asynchronous code in a more readable and concise way, while async/await is a more recent addition to JavaScript that makes working with promises even easier. For example:

```
// Using Promises
function fetchData() {
  return fetch("https://example.com/data")
    .then(response => response.json())
    .then(data => {
      // Do something with the data
    });
}

// Using Async/Await
async function fetchData() {
  const response = await
fetch("https://example.com/data");
  const data = await response.json();
  // Do something with the data
}
```

In conclusion, following these JavaScript best practices can lead to more efficient, maintainable, and reliable code. It is important to always strive for clean and readable code, and to continually update your skills as new best practices emerge.

JavaScript Questions and Answers

In JavaScript, the double equal sign "==" is known as the equality operator. It is used to compare two values and returns a Boolean value, either true or false, indicating whether the values are equal or not.

The comparison made by the double equal sign is based on the value of the operands, not on their data types. If the values being compared have different data types, JavaScript will try to convert one or both of the operands to the same data type before making the comparison.

Here are some examples of using the double equal sign in JavaScript:

```
console.log(5 == 5); // true
console.log(5 == '5'); // true
console.log(5 == 'hello'); // false
```

```
console.log(true == 1); // true
console.log(false == 0); // true
```

In the first example, the double equal sign is used to compare two numbers, both of which have the same value, so the comparison returns true.

In the second example, the double equal sign is used to compare a number and a string that represent the same value. JavaScript converts the string to a number before making the comparison, so the comparison returns true.

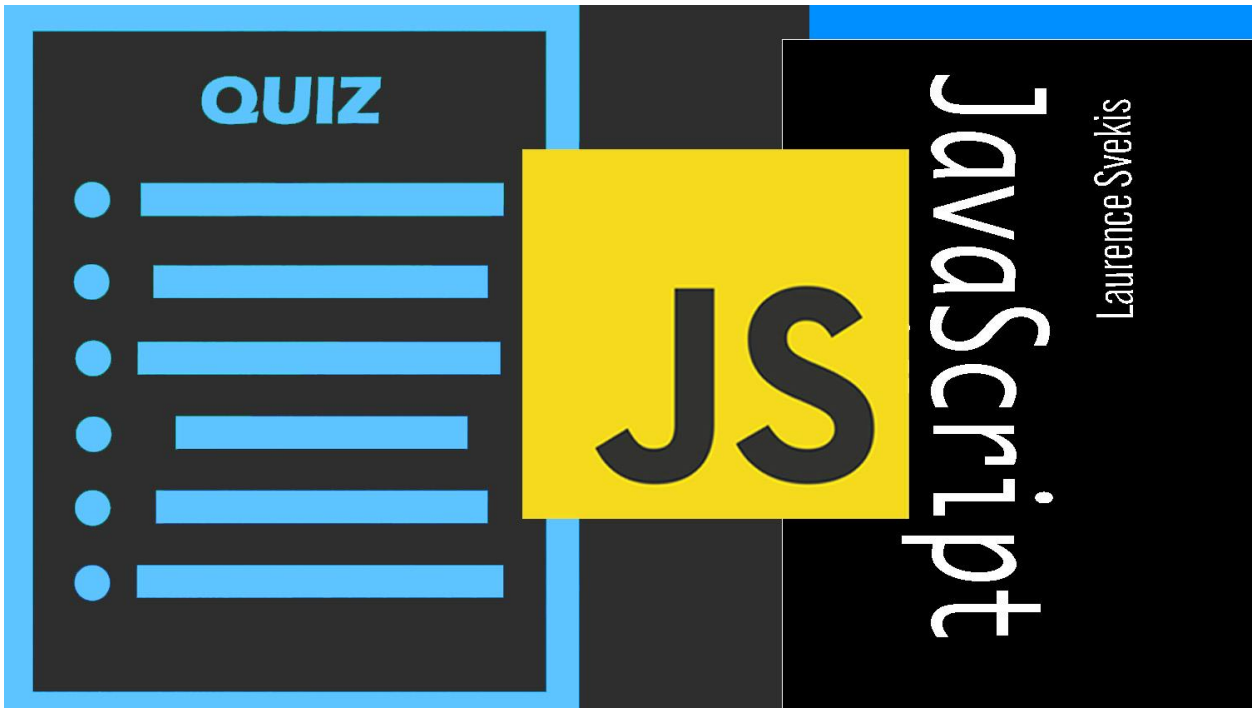
In the third example, the double equal sign is used to compare a number and a string that represent different values. Since the values are not equal, the comparison returns false.

In the fourth and fifth examples, the double equal sign is used to compare a boolean value and a number. JavaScript converts the boolean to a number (1 for true, 0 for false) before making the comparison, so both comparisons return true.

It's worth noting that using the double equal sign for comparisons can lead to unexpected results, especially when dealing with different data types. For this reason, it's generally recommended to use the triple equal sign "===" instead, which is known as the

strict equality operator. This operator not only compares the values, but also checks that the data types are the same.

JavaScript Quiz



What is the result of the following code?

```
console.log(1 + "2" + 3 + 4);
```

a) "1234"

b) "10"

c) "6"

d) "124"

What is the result of the following code?

```
console.log(typeof null);
```

a) "object"

b) "null"

c) "undefined"

d) "boolean"

Which of the following is NOT a valid way to declare a variable in JavaScript?

a) let myVar = "hello";

b) var myVar = "hello";

c) const myVar = "hello";

d) myVar = "hello";

What is the output of the following code?

```
let arr = [1, 2, 3, 4];
console.log(arr.slice(1, 3));
```

a) [2, 3]

b) [1, 2]

c) [3, 4]

d) [2, 4]

What is the result of the following code?

```
const a = 10;
a = 20;
console.log(a);
```

a) 10

b) 20

c) "const" keyword error

d) "let" keyword error

What is the result of the following code?

```
let num = 5;
num *= 10;
console.log(num);
```

a) 5

b) 10

c) 50

d) "num" keyword error

What is the result of the following code?

```
const obj = { name: "John", age: 30 };
console.log(obj.name);
```

a) "John"

b) "age"

c) { name: "John", age: 30 }

d) "undefined"

What is the output of the following code?

```
let arr = [1, 2, 3];
arr.push(4);
console.log(arr);
```

a) [1, 2, 3, 4]

b) [4, 3, 2, 1]

c) [1, 2, 3]

d) [1, 2, 4, 3]

Which of the following is NOT a valid data type in JavaScript?

a) string

b) boolean

c) number

d) float

What is the result of the following code?

console.log(5 == "5");

a) true

b) false

c) "5" keyword error

d) "==" keyword error

JavaScript ANSWERS

What is the result of the following code?

```
console.log(1 + "2" + 3 + 4);
```

Answer: a) "1234"

What is the result of the following code?

```
console.log(typeof null);
```

Answer: a) "object"

Which of the following is NOT a valid way to declare a variable in JavaScript?

Answer: d) myVar = "hello"; (this is an assignment statement, not a declaration)

What is the output of the following code?

```
let arr = [1, 2, 3, 4];
console.log(arr.slice(1, 3));
```

Answer: a) [2, 3]

What is the result of the following code?

```
const a = 10;
a = 20;
console.log(a);
```

Answer: c) "const" keyword error

What is the result of the following code?

```
let num = 5;
num *= 10;
console.log(num);
```

Answer: c) 50

What is the result of the following code?

```
const obj = { name: "John", age: 30 };
console.log(obj.name);
```

Answer: a) "John"

What is the output of the following code?

```
let arr = [1, 2, 3];
arr.push(4);
console.log(arr);
```

Answer: a) [1, 2, 3, 4]

Which of the following is NOT a valid data type in JavaScript?

Answer: d) float (JavaScript only has one numeric data type, which is "number")

What is the result of the following code?

console.log(5 == "5");

Answer: a) true (loose comparison between a number and a string will result in true if their values are equal, regardless of their data type)

JavaScript Code Mistakes and Solutions 4

Not using 'try-catch-finally' for resource management:

Not using 'try-catch-finally' can cause resource leaks and make it difficult to ensure resources are properly released. The solution is to use 'try-catch-finally' to ensure resources are always released.

```
let resource = null;
try {
  resource = acquireResource();
  // Use the resource
} catch (error) {
  console.log(error);
} finally {
  if (resource) {
    releaseResource(resource);
  }
}
```

Not using 'const' for values that won't change:

Not using 'const' for values that won't change can make your code less clear and make it easier to introduce bugs. The solution is to use 'const' for values that won't change.

```
const pi = 3.14159;
```

Not checking for null or undefined:

Not checking for null or undefined can lead to unexpected behavior and make it difficult to debug your code. The solution is to always check for null or undefined before using a variable or property.

```
if (myVariable !== null && myVariable !== undefined) {
  // Do something with myVariable
}
```

Not using 'Array.prototype.filter()':

Using 'for' loops to filter arrays can be tedious and error-prone. The solution is to use 'Array.prototype.filter()' to create a new array with only the elements that pass a test.

```
const newArray = array.filter((item) => {
  return item.passesTest();
});
```

Not using 'Array.prototype.forEach()':

Using 'for' loops to iterate over arrays and execute a function for each element can be tedious and error-prone. The solution is to use 'Array.prototype.forEach()' to execute a function for each element.

```
array.forEach((item) => {
  // Do something with item
});
```

Not using 'Object.assign()' for object cloning:

Using 'for' loops to clone objects can be tedious and error-prone. The solution is to use 'Object.assign()' to create a new object with the same properties as an existing object.

```
const newObj = Object.assign({}, oldObj);
```

Not using 'Array.prototype.some()' or 'Array.prototype.every()':

Using 'for' loops to check if any or all elements in an array pass a test can be tedious and error-prone. The solution is to use 'Array.prototype.some()' or 'Array.prototype.every()' to check if any or all elements in an array pass a test.

```
if (array.some((item) => item.passesTest())) {
  // At least one item passes the test
}

if (array.every((item) => item.passesTest())) {
  // All items pass the test
}
```

Not using 'Array.prototype.find()' or 'Array.prototype.findIndex()':

Using 'for' loops to find the first element that passes a test or its index can be tedious and error-prone. The solution is to use 'Array.prototype.find()' or 'Array.prototype.findIndex()' to find the first element that passes a test or its index.

```
const foundItem = array.find((item) =>
item.passesTest());
const foundIndex = array.findIndex((item) =>
item.passesTest());
```

Not using 'Array.prototype.sort()':

Using 'for' loops to sort arrays can be tedious and error-prone. The solution is to use 'Array.prototype.sort()' to sort arrays.

Not handling asynchronous code properly:

Not handling asynchronous code properly can lead to unexpected behavior and make it difficult to debug your code. The solution is to use promises, async/await, or callbacks to handle asynchronous code.

```
// Using promises
myPromise.then((result) => {
  // Do something with result
}).catch((error) => {
  console.log(error);
});

// Using async/await
async function myFunction() {
  try {
    const result = await myPromise;
    // Do something with result
  } catch (error) {
    console.log(error);
  }
}

// Using callbacks
myFunction((error, result) => {
  if (error) {
    console.log(error);
  } else {
    // Do something with result
  }
});
```

By avoiding these common mistakes, you can write more efficient, effective, and maintainable JavaScript code.

JavaScript Code Exercises 1

[Write a function that takes an array of numbers and returns the sum of all the positive numbers in the array.]

[Write a function that takes an array of strings and returns the shortest string in the array.]

[Write a function that takes a string and returns a new string with all the vowels removed.]

[Write a function that takes an array of numbers and returns a new array with all the even numbers removed.]

Write a function that takes an array of numbers and returns the sum of all the positive numbers in the array.

```
function sumOfPositiveNumbers(numbers) {
  let sum = 0;
  for (let i = 0; i < numbers.length; i++) {
    if (numbers[i] > 0) {
      sum += numbers[i];
    }
  }
  return sum;
}
```

Explanation: This function iterates over the array using a for loop and adds up all the positive numbers it encounters. The sum is stored in the sum variable and is returned at the end.

Write a function that takes an array of strings and returns the shortest string in the array.

```
function findShortestString(strings) {
  let shortest = strings[0];
  for (let i = 1; i < strings.length; i++) {
    if (strings[i].length < shortest.length) {
      shortest = strings[i];
    }
  }
  return shortest;
```

```
}
```

Explanation: This function uses a for loop to iterate over the array and keep track of the shortest string it has encountered so far. The shortest variable is initialized to the first string in the array, and is updated whenever a shorter string is found.

Write a function that takes a string and returns a new string with all the vowels removed.

```
function removeVowels(str) {
  return str.replace(/[aeiou]/gi, '');
}
```

Explanation: This function uses the replace method on the string to remove all occurrences of the vowels (both upper and lowercase) using a regular expression.

Write a function that takes an array of numbers and returns a new array with all the even numbers removed.

```
function removeEvenNumbers(numbers) {
  return numbers.filter(num => num % 2 !== 0);
}
```

Explanation: This function uses the filter method on the array to create a new array with only the odd numbers

JavaScript Code Exercises 3

Exercise 1: Title case a sentence

Write a function that takes a sentence as input and returns a new sentence where the first letter of each word is capitalized.

```
function titleCase(str) {
  const words = str.toLowerCase().split(" ");
  for (let i = 0; i < words.length; i++) {
```

```
    words[i] = words[i][0].toUpperCase() +
words[i].slice(1);
  }
  return words.join(" ");
}

console.log(titleCase("hello world")); // Output:
"Hello World"
```

In this example, we first convert the input string to lowercase and split it into an array of words. We then use a for loop to capitalize the first letter of each word by taking the first letter of the word, converting it to uppercase, and then appending the rest of the word. Finally, we join the array of words back into a string with spaces between them.

Exercise 2: Find the largest number in an array

Write a function that takes an array of numbers as input and returns the largest number in the array.

```
function largestNumber(arr) {
  let max = arr[0];
  for (let num of arr) {
    if (num > max) {
      max = num;
    }
  }
  return max;
}
```

```
console.log(largestNumber([1, 2, 3, 4, 5])); // Output:
5
```

In this example, we initialize a variable called max to the first element in the array. We then use a for loop to iterate through the rest of the array, updating the value of max if we find a larger number. Finally, we return the largest number we found.

Exercise 3: Confirm all elements in an array are the same

Write a function that takes an array as input and returns true if all the elements in the array are the same, and false otherwise.

```
function allEqual(arr) {
  for (let i = 1; i < arr.length; i++) {
    if (arr[i] !== arr[0]) {
      return false;
    }
  }
  return true;
}

console.log(allEqual([1, 1, 1, 1])); // Output: true
console.log(allEqual([1, 2, 3, 4])); // Output: false
```

In this example, we use a for loop to compare each element in the array to the first element. If we find any element that is not equal to the first element, we return false. If we make it through the entire loop without finding any unequal elements, we return true.

Exercise 4: Count the number of vowels in a string

Write a function that takes a string as input and returns the number of vowels (a, e, i, o, u) in the string.

```
function countVowels(str) {
  const vowels = "aeiou";
  let count = 0;
  for (let letter of str.toLowerCase()) {
    if (vowels.includes(letter)) {
      count++;
    }
  }
  return count;
}

console.log(countVowels("hello world")); // Output: 3
```

In this example, we first initialize a variable called vowels to a string containing all the vowels. We also initialize a count variable to 0. We then use a for loop to iterate through each letter in the input string, converting it to lowercase. If the letter is a vowel,

JavaScript Code Exercises 2

Exercise 1: Reverse a string

Write a function that takes a string as input and returns the string reversed.

```
function reverseString(str) {
  return str.split("").reverse().join("");
}

console.log(reverseString("hello")); // Output: "olleh"
```

In this example, we first split the string into an array of characters, then we reverse the array, and finally we join the reversed array back into a string.

Exercise 2: Factorialize a number

Write a function that takes a number as input and returns the factorial of that number.

javascript
Copy code

```
function factorialize(num) {
  if (num === 0) {
    return 1;
  } else {
    return num * factorialize(num - 1);
  }
}

console.log(factorialize(5)); // Output: 120
```

In this example, we use recursion to calculate the factorial of a number. If the input number is 0, we return 1. Otherwise, we call the function again with num - 1 as input and multiply it with the current value of num.

Exercise 3: Check for palindrome

Write a function that takes a string as input and returns true if the string is a palindrome and false otherwise.

```
function isPalindrome(str) {
  const reversedStr = str.split("").reverse().join("");
  return str === reversedStr;
}

console.log(isPalindrome("racecar")); // Output: true
console.log(isPalindrome("hello")); // Output: false
```

In this example, we first reverse the input string and store it in a new variable called reversedStr. We then compare the original string to the reversed string and return true if they are equal.

Exercise 4: Find the longest word

Write a function that takes a string as input and returns the length of the longest word in the string.

```
function findLongestWord(str) {
  const words = str.split(" ");
  let longestWord = "";
  for (let word of words) {
    if (word.length > longestWord.length) {
      longestWord = word;
    }
  }
  return longestWord.length;
}
```

```
console.log(findLongestWord("The quick brown fox jumped
over the lazy dog")); // Output: 6
```

In this example, we first split the input string into an array of words. We then loop through each word and compare its length to the length of the current longest word. If the current word is longer, we update the value of longestWord. Finally, we return the length of the longest word.

Exercise 5: Title case a sentence

Write a function that takes a string as input and returns the string with the first letter of each word capitalized.

```
function titleCase(str) {
  const words = str.toLowerCase().split(" ");
  const titleCasedWords = [];
  for (let word of words) {
    titleCasedWords.push(word[0].toUpperCase() +
word.slice(1));
  }
  return titleCasedWords.join(" ");
}

console.log(titleCase("the quick brown fox")); //
Output: "The Quick Brown Fox"
```

In this example, we first convert the input string to lowercase and split it into an array of words. We then loop through each word, capitalize the first letter, and add it to a new array called titleCasedWords. Finally, we join the words in titleCasedWords back into a string and return it.

Exercise 6: Return largest numbers in arrays

Write a function that takes an array of arrays of numbers as input and returns an array containing the largest number from each array.

```
function largestNumbers(arr) {
  const largest = [];
  for (let subArr of arr) {
    let max = subArr[0];
    for (let num of subArr) {
      if (num > max) {
        max = num;
      }
    }
    largest.push(max);
  }
  return largest;
}

console.log(largestNumbers([[1, 2, 3], [4, 5, 6], [7,
8, 9]])); // Output: [3, 6, 9]
```

In this example, we first create an empty array called largest. We then loop through each sub-array in the input array and find the largest number in that sub-array using another for loop. We update the value of max if we find a larger number. Finally, we add the largest number to the largest array and return it.

Exercise 7: Confirm the ending

Write a function that takes a string and a target substring as input and returns true if the string ends with the target substring, and false otherwise.

```
function confirmEnding(str, target) {
  return str.slice(-target.length) === target;
}

console.log(confirmEnding("hello world", "world")); //
Output: true
console.log(confirmEnding("hello world", "goodbye"));
// Output: false
```

In this example, we use the slice() method to extract the substring of str that is the same length as target, starting from the end of the string. We then compare this extracted substring to the target substring and return true if they are equal.

Exercise 8: Repeat a string

Write a function that takes a string and a number as input and returns a new string that repeats the input string a given number of times.

```
function repeatString(str, num) {
  let repeated = "";
  for (let i = 0; i < num; i++) {
    repeated += str;
  }
  return repeated;
}
```

```
console.log(repeatString("abc", 3)); // Output: "abcabcabc"
```

In this example, we create an empty string called repeated. We then use a for loop to repeat the input string num times by adding it to the repeated string each iteration. Finally, we return the repeated string.

Exercise 9: Truncate a string

Write a function that takes a string and a number as input and returns a truncated version of the string, ending with an ellipsis (...) if the original string was longer than the given number.

```
function truncateString(str, num) {
  if (str.length <= num) {
    return str;
  } else {
    return str.slice(0, num) + "...";
  }
}

console.log(truncateString("hello world", 5)); // Output: "hello..."
```

In this example, we first check if the length of the input string is less than or equal to the given number. If it is, we simply return the original string. Otherwise, we use the slice() method to extract the first num characters of the string, and append an ellipsis to the end.

Exercise 10: Chunky monkey

Write a function that takes an array and a number as input and returns an array of arrays, where each sub-array contains a maximum of the given number of elements from the original array.

```
function chunkArray(arr, size) {
  const chunks = [];
  let i = 0;
  while (i < arr.length) {
    chunks.push(arr.slice(i, i + size));
    i += size;
  }
  return chunks;
}

console.log(chunkArray([1, 2, 3, 4, 5], 2)); // Output:
[[1, 2], [3, 4], [5]]
```

In this example, we create an empty array called chunks. We then use a while loop to iterate through the input array, taking slices of size elements at a time and pushing them to the chunks array. We use a separate variable i to keep track of our position in the input array. Finally, we return the chunks array.

JavaScript Code Exercises 4

Exercise 1: Sum All Numbers in a Range

Write a function that takes an array of two numbers as input and returns the sum of all the numbers between them, inclusive.

```
function sumAll(arr) {
  let min = Math.min(...arr);
  let max = Math.max(...arr);
  let sum = 0;
  for (let i = min; i <= max; i++) {
```

```
    sum += i;
  }
  return sum;
}

console.log(sumAll([1, 4])); // Output: 10
```

In this example, we first use the Math.min() and Math.max() functions to find the minimum and maximum values in the input array, respectively. We then use a for loop to iterate from the minimum value to the maximum value, adding each number to the sum variable. Finally, we return the sum.

Exercise 2: Diff Two Arrays

Write a function that takes two arrays as input and returns a new array containing the elements that are unique to each array.

```
function diffArray(arr1, arr2) {
  return arr1.filter((item) => !arr2.includes(item))
    .concat(arr2.filter((item) =>
!arr1.includes(item)));
}

console.log(diffArray([1, 2, 3, 4], [3, 4, 5, 6])); //
Output: [1, 2, 5, 6]
```

In this example, we first use the filter() method to create a new array that contains only the elements from arr1 that are not in arr2. We then concatenate this array with a similar array created from arr2. The filter() method with the includes() method inside of it allows us to filter out the duplicate elements between both arrays. Finally, we return the concatenated array.

Exercise 3: Seek and Destroy

Write a function that takes an arbitrary number of arguments, the first of which is an array, and removes all other arguments from the array.

```
function destroyer(arr, ...args) {
  return arr.filter((item) => !args.includes(item));
}

console.log(destroyer([1, 2, 3, 4, 5], 2, 3)); //
Output: [1, 4, 5]
```

In this example, we use the rest parameter syntax to allow the function to accept an arbitrary number of arguments. We then use the filter() method to create a new array that contains only the elements from arr that are not in args. Finally, we return the filtered array.

Exercise 4: Wherefore art thou

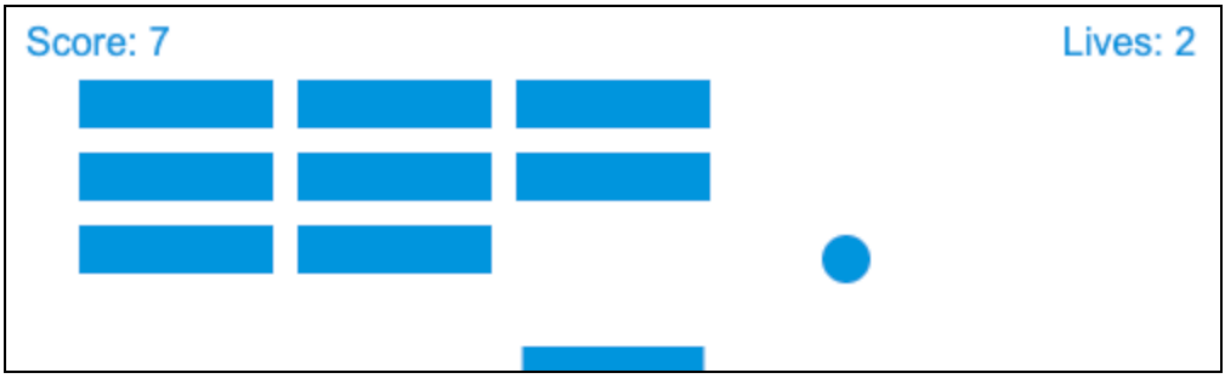
Write a function that takes two arrays of objects as input, and returns an array of all the objects in the first array that have matching property-value pairs to the objects in the second array.

```
function whatIsInAName(collection, source) {
  return collection.filter((obj) => {
    for (let key in source) {
      if (!obj.hasOwnProperty(key) || obj[key] !==
source[key]) {
        return false;
      }
    }
```

```
    return true;
  });
}

console.log(whatIsInAName([{name: "John", age: 25},
{name: "Jane", age: 27}], {age: 25})); // Output:
[{name: "John", age: 25}]
```

In this example, we use the filter() method to create a new array that contains only the objects from collection that match the property-value pairs in source. We loop through each key in source and check if the object in collection has that key

JavaScript BreakOut Game



This code is an HTML file that contains a canvas element with an id of "canvas" and some JavaScript code to create a game of Brick Breaker. The canvas element is used to display the game graphics and the JavaScript code is used to handle user input, move the game elements, and detect collisions between the ball and the bricks.

The JavaScript code initializes variables for the game elements, including the ball, paddle, and bricks, and adds event listeners to handle user input from the keyboard and mouse. There are functions to draw the game elements on the canvas, including the

ball, paddle, bricks, score, and lives. There is also a function to detect collisions between the ball and the bricks.

The game loop is handled by the draw function, which is called repeatedly using requestAnimationFrame. This function updates the position of the ball, checks for collisions, and redraws the game elements on the canvas. If the player breaks all the bricks, they win the game. If the player loses all their lives, they lose the game. The game can be restarted by reloading the page.

```
<!DOCTYPE html>
<html>

<head>
  <title>Nav Menu</title>
  <style>
    canvas {
      border: 1px solid black;
    }
  </style>
</head>

<body>
  <canvas id="canvas" width="500px"></canvas>
  <script>
    // Initialize variables
    var canvas = document.getElementById("canvas");
    var ctx = canvas.getContext("2d");
    var x = canvas.width / 2;
    var y = canvas.height - 30;
```

```
var dx = 2;
var dy = -2;
var ballRadius = 10;
var paddleHeight = 10;
var paddleWidth = 75;
var paddleX = (canvas.width - paddleWidth) / 2;
var rightPressed = false;
var leftPressed = false;
var brickRowCount = 3;
var brickColumnCount = 5;
var brickWidth = 80;
var brickHeight = 20;
var brickPadding = 10;
var brickOffsetTop = 30;
var brickOffsetLeft = 30;
var score = 0;
var lives = 3;
var bricks = [];
for (var c = 0; c < brickColumnCount; c++) {
    bricks[c] = [];
    for (var r = 0; r < brickRowCount; r++) {
        bricks[c][r] = { x: 0, y: 0, status: 1 };
    }
}

// Add event listeners
document.addEventListener("keydown", keyDownHandler, false);
document.addEventListener("keyup", keyUpHandler, false);
document.addEventListener("mousemove", mouseMoveHandler, false);
```

```
// Functions to handle events
function keyDownHandler(e) {
    if (e.keyCode == 39) {
        rightPressed = true;
    }
    else if (e.keyCode == 37) {
        leftPressed = true;
    }
}

function keyUpHandler(e) {
    if (e.keyCode == 39) {
        rightPressed = false;
    }
    else if (e.keyCode == 37) {
        leftPressed = false;
    }
}

function mouseMoveHandler(e) {
    var relativeX = e.clientX - canvas.offsetLeft;
    if (relativeX > 0 && relativeX < canvas.width) {
        paddleX = relativeX - paddleWidth / 2;
    }
}

function collisionDetection() {
    for (var c = 0; c < brickColumnCount; c++) {
```

```
            for (var r = 0; r < brickRowCount; r++) {
                var b = bricks[c][r];
                if (b.status == 1) {
                    if (x > b.x && x < b.x + brickWidth && y > b.y && y < b.y +
brickHeight) {
                        dy = -dy;
                        b.status = 0;
                        score++;
                        if (score == brickRowCount * brickColumnCount) {
                            alert("You win!");
                            document.location.reload();
                        }
                    }
                }
            }
        }
    }

    // Draw functions
    function drawBall() {
        ctx.beginPath();
        ctx.arc(x, y, ballRadius, 0, Math.PI * 2);
        ctx.fillStyle = "#0095DD";
        ctx.fill();
        ctx.closePath();
    }

    function drawPaddle() {
        ctx.beginPath();
```

```
        ctx.rect(paddleX, canvas.height - paddleHeight, paddleWidth,
paddleHeight);
        ctx.fillStyle = "#0095DD";
        ctx.fill();
        ctx.closePath();
    }
    // Draw functions (continued)
    function drawBricks() {
        for (var c = 0; c < brickColumnCount; c++) {
            for (var r = 0; r < brickRowCount; r++) {
                if (bricks[c][r].status == 1) {
                    var brickX = (c * (brickWidth + brickPadding)) +
brickOffsetLeft;
                    var brickY = (r * (brickHeight + brickPadding)) +
brickOffsetTop;
                    bricks[c][r].x = brickX;
                    bricks[c][r].y = brickY;
                    ctx.beginPath();
                    ctx.rect(brickX, brickY, brickWidth, brickHeight);
                    ctx.fillStyle = "#0095DD";
                    ctx.fill();
                    ctx.closePath();
                }
            }
        }
    }

    function drawScore() {
        ctx.font = "16px Arial";
```

```
    ctx.fillStyle = "#0095DD";
    ctx.fillText("Score: " + score, 8, 20);
}

function drawLives() {
    ctx.font = "16px Arial";
    ctx.fillStyle = "#0095DD";
    ctx.fillText("Lives: " + lives, canvas.width - 65, 20);
}

function draw() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    drawBricks();
    drawBall();
    drawPaddle();
    drawScore();
    drawLives();
    collisionDetection();

    // Bounce ball off walls and paddle
    if (x + dx > canvas.width - ballRadius || x + dx < ballRadius) {
        dx = -dx;
    }
    if (y + dy < ballRadius) {
        dy = -dy;
    }
    else if (y + dy > canvas.height - ballRadius) {
        if (x > paddleX && x < paddleX + paddleWidth) {
            dy = -dy;
```

```
        }
        else {
            lives--;
            if (!lives) {
                alert("Game over!");
                document.location.reload();
            }
            else {
                x = canvas.width / 2;
                y = canvas.height - 30;
                dx = 2;
                dy = -2;
                paddleX = (canvas.width - paddleWidth) / 2;
            }
        }
    }

    // Move paddle
    if (rightPressed && paddleX < canvas.width - paddleWidth) {
        paddleX += 7;
    }
    else if (leftPressed && paddleX > 0) {
        paddleX -= 7;
    }

    x += dx;
    y += dy;
    requestAnimationFrame(draw);
}
```

```
    draw();
  </script>
</body>

</html>
```

Explanation:

The code creates a simple game where the player has to use the paddle to bounce the ball and break the bricks. The game consists of an HTML canvas element where the graphics are drawn using JavaScript.

The code initializes all the variables needed for the game, including the canvas, the ball, the paddle, the bricks, the score, and the lives. It also adds event listeners for keyboard and mouse input to move the paddle.

The code defines several functions to draw the game elements on the canvas, including the ball, the paddle, the bricks, the score, and the lives. It also defines a function to handle collisions between the ball and the bricks.

The main game loop is implemented in the `draw()` function, which is called repeatedly using the `requestAnimationFrame()` method. The function clears the canvas, draws the game elements, checks for collisions, and updates the position of the ball and the paddle.

The game ends when the player runs out of lives or breaks all the bricks. In either case, an alert is shown to notify the player, and the game is reset by reloading the page.

5 JavaScript coding Exercises 2

FizzBuzz

Source code:

```
for (let i = 1; i <= 100; i++) {
  if (i % 3 === 0 && i % 5 === 0) {
    console.log('FizzBuzz');
  } else if (i % 3 === 0) {
    console.log('Fizz');
  } else if (i % 5 === 0) {
    console.log('Buzz');
  } else {
    console.log(i);
  }
}
```

Explanation:
The FizzBuzz exercise involves iterating from 1 to 100 and printing out "Fizz" for multiples of 3, "Buzz" for multiples of 5, and "FizzBuzz" for multiples of both 3 and 5. We can use the

modulus operator to check if a number is a multiple of another number, and use if/else statements to determine what to print.

Palindrome checker

Source code:

```
function isPalindrome(str) {
  const len = str.length;
  for (let i = 0; i < len / 2; i++) {
    if (str[i] !== str[len - 1 - i]) {
      return false;
    }
  }
  return true;
}

console.log(isPalindrome('racecar')); // true
console.log(isPalindrome('hello')); // false
```

Explanation:

The palindrome checker exercise involves writing a function that checks if a given string is a palindrome. We can iterate over the first half of the string and compare each character to the corresponding character from the end of the string. If any characters don't match, we can return false. If we get through the entire loop without finding any mismatches, we can return true.

Hangman game

Source code:

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

function getRandomWord() {
  const index = Math.floor(Math.random() *
words.length);
  return words[index];
}

function hideWord(word) {
  let hiddenWord = '';
  for (let i = 0; i < word.length; i++) {
    hiddenWord += '_';
  }
  return hiddenWord;
}

function playHangman() {
  const word = getRandomWord();
  let hiddenWord = hideWord(word);
  let remainingGuesses = 6;
  while (remainingGuesses > 0) {
    console.log(`Word: ${hiddenWord}`);
    console.log(`Guesses remaining:
${remainingGuesses}`);
    const guess = prompt('Guess a letter:');
    if (!word.includes(guess)) {
      remainingGuesses--;
    }
    for (let i = 0; i < word.length; i++) {
      if (word[i] === guess) {
```

```
        hiddenWord = hiddenWord.slice(0, i) + guess +
hiddenWord.slice(i + 1);
      }
    }
    if (hiddenWord === word) {
      console.log(`You win! The word was ${word}`);
      return;
    }
  }
  console.log(`You lose! The word was ${word}`);
}

playHangman();
```

Explanation:
The hangman game exercise involves creating a simple game where the computer chooses a random word and the player has to guess the letters of the word before running out of guesses. We can use an array of words to choose from, and prompt the user for their guesses. We can keep track of the remaining guesses and update the hidden word as the user guesses correct letters.

Shopping cart

Source code:

```
const products = [  { name: 'Apple', price: 1.99 },  {
name: 'Banana', price: 0.99 },  { name: 'Cherry',
price: 2.99 },  { name: 'Dragonfruit', price: 4.99 },
{ name: 'Elderberry', price: 3.99 },];
```

```
function addToCart(cart, product) {
  cart.push(product);
}

function removeFromCart(cart, productName) {
  const index = cart.findIndex(item => item.name ===
productName);
  if (index >= 0) {
    cart.splice(index, 1);
  }
}

function calculateTotal(cart) {
  let total = 0;
  for (let i = 0; i < cart.length; i++) {
    total += cart[i].price;
  }
  return total.toFixed(2);
}

const shoppingCart = [];
addToCart(shoppingCart, products[0]);
addToCart(shoppingCart, products[2]);
addToCart(shoppingCart, products[4]);
console.log(shoppingCart);
removeFromCart(shoppingCart, 'Banana');
console.log(shoppingCart);
console.log(`Total: $${calculateTotal(shoppingCart)}`);
```

Explanation:

The shopping cart exercise involves creating a program that allows the user to add and remove items from their cart, calculate the total price, and apply discounts or coupons. We can use an

array to represent the shopping cart, and objects to represent the products. We can define functions for adding and removing items from the cart, and for calculating the total price by iterating over the cart array and adding up the prices of the items. We can also use the toFixed method to ensure that the total is displayed with two decimal places.

Typing speed test

Source code:

```
const paragraph = 'The quick brown fox jumps over the
lazy dog';
let startTime;
let endTime;

function startTest() {
  startTime = Date.now();
  document.getElementById('prompt').innerHTML =
paragraph;

document.getElementById('input').addEventListener('inpu
t', checkInput);
}

function checkInput() {
  const typedText =
document.getElementById('input').value;
  if (typedText === paragraph) {
    endTime = Date.now();
    const totalTime = (endTime - startTime) / 1000;
```

```
    const speed = Math.round(paragraph.length /
totalTime);
    document.getElementById('result').innerHTML = `You
typed at a speed of ${speed} characters per second!`;

document.getElementById('input').removeEventListener('i
nput', checkInput);
  }
}

document.getElementById('start-button').addEventListene
r('click', startTest);
```

Explanation:
The typing speed test exercise involves creating a program that measures the user's typing speed by having them type a given paragraph as quickly and accurately as possible. We can use HTML and CSS to create a simple user interface with a start button, a prompt to type, an input field to type into, and a result section to display the user's speed. We can use the Date.now() method to record the start and end times of the test, and the addEventListener method to listen for input events in the input field. We can then compare the typed text to the prompt text and calculate the speed by dividing the length of the prompt by the total time in seconds. Finally, we can update the result section with the user's speed and remove the event listener to stop listening for input events.

10 JavaScript Coding Mistakes with solutions

Not declaring variables properly:

One of the most common mistakes in JavaScript is forgetting to declare a variable using the "var", "let", or "const" keyword. This can lead to unexpected results and make debugging difficult.

Example:

```
x = 10; // This variable is not declared
console.log(x); // Output: 10
```

Instead, declare the variable using "var", "let", or "const":

```
let x = 10;
console.log(x); // Output: 10
```

Using the wrong comparison operator:

Using the wrong comparison operator can lead to unexpected results, especially when comparing strings and numbers. The "===" operator checks for both value and data type equality, while the "==" operator only checks for value equality.

Example:

```
console.log(1 == '1'); // Output: true
console.log(1 === '1'); // Output: false
```

Not using semicolons:

JavaScript uses semicolons to separate statements. While the interpreter will automatically insert them in most cases, it's best practice to include them yourself to avoid unexpected results.

Example:

```
let x = 10
let y = 20
console.log(x + y) // Output: 30
```

Instead, add semicolons:

```
let x = 10;
let y = 20;
console.log(x + y); // Output: 30
```

Not understanding scoping:

Variables declared inside a function have local scope and are not accessible outside of the function. Variables declared outside of a function have global scope and can be accessed from anywhere in the code.

Example:

```
let x = 10;

function myFunction() {
  let x = 5;
  console.log(x); // Output: 5
```

```
}

myFunction();
console.log(x); // Output: 10
```

Using "var" instead of "let" or "const":

"var" declares a variable with function scope, while "let" and "const" declare variables with block scope. Using "var" can lead to unexpected results when trying to access variables outside of their intended scope.

Example:

```
function myFunction() {
  if (true) {
    var x = 5;
  }
  console.log(x); // Output: 5
}

myFunction();
console.log(x); // Output: ReferenceError: x is not
defined
```

Instead, use "let" or "const":

```
function myFunction() {
  if (true) {
    let x = 5;
  }
  console.log(x); // Output: ReferenceError: x is not
defined
}

myFunction();
console.log(x); // Output: ReferenceError: x is not
defined
```

Using "==" instead of "===":

As mentioned earlier, "==" only checks for value equality, while "===" checks for both value and data type equality. Using "==" can lead to unexpected results.

Example:

```
console.log(1 == '1'); // Output: true
Instead, use "===":
console.log(1 === '1'); // Output: false
```

Not using curly braces in if statements:

While it's possible to omit curly braces in if statements when there's only one statement, it's best practice to include them to avoid confusion.

Example:

```
if (x > 10)
  console.log('x is greater than 10');
```

Instead, use curly braces:

```
if (x > 10) {
  console.log('x is greater than 10');
}
```

Using "parseFloat" instead of "parseInt":

"parseFloat" is used to convert a string to a floating-point number, while "parseInt" is used to convert a string to an integer. Using the wrong method can lead to unexpected results.

Example:

```
console.log(parseFloat('10.5')); // Output: 10.5
console.log(parseInt('10.5')); // Output: 10
```

Not handling asynchronous code properly:

JavaScript is a single-threaded language, but it can handle asynchronous tasks using callbacks, promises, and async/await. Not handling asynchronous code properly can lead to race conditions and other bugs.
Example using a callback:

```
function myFunction(callback) {
  setTimeout(function() {
    callback();
  }, 1000);
}

myFunction(function() {
  console.log('Callback executed after 1 second');
});
```

Example using a promise:

```
function myFunction() {
  return new Promise(function(resolve, reject) {
    setTimeout(function() {
      resolve('Promise resolved after 1 second');
    }, 1000);
  });
}
```

```
myFunction().then(function(result) {
  console.log(result);
});
```

Example using async/await:

```
async function myFunction() {
  try {
    const result = await new Promise(function(resolve,
reject) {
      setTimeout(function() {
        resolve('Async/await resolved after 1 second');
      }, 1000);
    });
    console.log(result);
  } catch (error) {
    console.error(error);
  }
}
myFunction();
```

Not properly scoping variables:

When declaring variables, it's important to declare them within the proper scope, otherwise they may not be accessible where they need to be.
Example:

```
function myFunction() {
  var x = 1; // x is declared in the function scope
  if (true) {
```

```
    var x = 2; // x is now 2, and it's also in the
function scope
  }
  console.log(x); // Output: 2
}
myFunction();
```

To properly scope variables, you can use the "let" and "const" keywords introduced in ES6.

Example using let:

```
function myFunction() {
  let x = 1; // x is declared in the function scope
  if (true) {
    let x = 2; // x is now 2, but it's only in the
block scope
  }
  console.log(x); // Output: 1
}
myFunction();
```

Example using var:

```
function myFunction() {
  var x = 1; // x is declared in the function scope
  if (true) {
    var x = 2; // x is now 2, and it's also in the
function scope
  }
  console.log(x); // Output: 2
}
myFunction();
```

In this example, x is declared using the var keyword inside the function myFunction(). When the if block is executed, x is assigned a new value of 2, which changes the value of x outside the if block as well. This is because variables declared with var have function-level scope, meaning they are accessible throughout the entire function, even if they are declared inside a block.

To properly scope variables, you can use the let and const keywords introduced in ES6. Here's an example:

```
function myFunction() {
  let x = 1; // x is declared in the function scope
  if (true) {
    let x = 2; // x is now 2, but it's only in the
block scope
  }
  console.log(x); // Output: 1
}
myFunction();
```

In this example, x is declared using the let keyword inside the function myFunction(). When the if block is executed, a new variable called x is created with a value of 2, but this variable is only accessible inside the block. The original variable x with a value of 1 is still accessible outside the block and is outputted to the console.

5 JavaScript coding Exercises 1

Reverse a String:

Write a function that takes a string as an argument and returns the string in reverse order. For example, if the input string is "hello", the output should be "olleh".

```
function reverseString(str) {
  return str.split("").reverse().join("");
}

console.log(reverseString("hello"));
```

FizzBuzz:

Write a function that takes an integer n as an argument and prints out numbers from 1 to n. For multiples of 3, print "Fizz" instead of the number. For multiples of 5, print "Buzz". For multiples of both 3 and 5, print "FizzBuzz".

```
function fizzBuzz(n) {
  for (let i = 1; i <= n; i++) {
    if (i % 3 === 0 && i % 5 === 0) {
      console.log("FizzBuzz");
    } else if (i % 3 === 0) {
```

```
      console.log("Fizz");
    } else if (i % 5 === 0) {
      console.log("Buzz");
    } else {
      console.log(i);
    }
  }
}

fizzBuzz(15);
```

Palindrome Checker:

Write a function that takes a string as an argument and returns true if the string is a palindrome (reads the same backward as forward), otherwise false.

```
function isPalindrome(str) {
  const reversedStr = str.split("").reverse().join("");
  return str === reversedStr;
}

console.log(isPalindrome("racecar"));
console.log(isPalindrome("hello"));
```

Random Number Generator:

Write a function that generates a random number between 0 and a given maximum value (inclusive).

```
function getRandomNumber(max) {
  return Math.floor(Math.random() * (max + 1));
}
```

```
console.log(getRandomNumber(10));
```

Capitalize the First Letter of Each Word:

Write a function that takes a string as an argument and returns the same string with the first letter of each word capitalized.

```
function capitalizeWords(str) {
  return str.split(" ").map(word =>
word.charAt(0).toUpperCase() + word.slice(1)).join("
");
}

console.log(capitalizeWords("hello world"));
```