JavaScript assignment

Q1: What is JavaScript?

Ans: JavaScript is a high-level, versatile, and widely-used programming language primarily employed for creating interactive and dynamic content on websites. It allows developers to add functionality to web pages, handle user interactions, and manipulate the Document Object Model (DOM) to modify the content and structure of web pages in real time.

Key characteristics of JavaScript include:

1. **Client-Side Scripting:** JavaScript is executed on the user's web browser, making it a client-side scripting language. This enables it to react to user actions without needing to send requests to the server, resulting in more responsive and interactive web applications.
2. **Versatile:** JavaScript is a versatile language that can be used for various purposes, including web development, server-side programming (Node.js), game development, mobile app development, and more.
3. **Interactivity:** JavaScript is essential for creating interactive features like form validation, animations, and dynamic content updates without requiring the user to refresh the entire page.
4. **DOM Manipulation:** It allows developers to manipulate the Document Object Model (DOM) of a web page. This means you can add, remove, or modify elements on a web page, changing its structure and appearance dynamically.
5. **Event Handling:** JavaScript is used to respond to events like user clicks, key presses, and mouse movements, enabling developers to create responsive user interfaces.
6. **Asynchronous Programming:** JavaScript supports asynchronous programming, allowing you to perform tasks such as making network requests without blocking the main execution thread, ensuring a smooth user experience.
7. **Open Standards:** JavaScript is based on open standards and is supported by all major web browsers, making it a fundamental technology for web development.

JavaScript is often used in conjunction with HTML and CSS to create modern, feature-rich web applications. It has a large and active developer community, which has led to the development of numerous libraries and frameworks, such as React, Angular, and Vue.js, that simplify web application development.

Q2: What is the use of the isNaN function?

Ans. The `isNaN` function in JavaScript is used to determine whether a given value is "Not-a-Number" (NaN) or not. NaN is a special value in JavaScript that represents the result of an operation that should return a valid number but doesn't.

The `isNaN` function takes one argument, which can be any value, and returns a Boolean value:

- If the argument is NaN (i.e., not a valid number), the function returns `true`.
- If the argument is a valid number or a value that can be converted to a valid number, the function returns `false`.

Here's how you can use the `isNaN` function:

javascriptCopy code

```javascript
isNaN(123); // Returns false, because 123 is a valid number. isNaN("Hello"); // Returns true, because "Hello" cannot be converted to a valid number. isNaN("123"); // Returns false, because "123" can be converted to a valid number. isNaN(NaN); // Returns true, because NaN is, by definition, "Not-a-Number."
```

It's important to note that `isNaN` has some quirks. For example, if you pass it a non-numeric string that can be implicitly converted to a number, it may return `false`. To address this, you can use `parseFloat` or `parseInt` to explicitly convert a string to a number and then check if it's `NaN` or not.

javascriptCopy code

```javascript
isNaN(parseFloat("123")); // Returns false, because "123" is converted to a valid number.
```

In modern JavaScript, you can also use the `Number.isNaN` method, which is more strict in its behavior and only returns `true` for values that are exactly NaN. It doesn't perform implicit type conversion.

javascriptCopy code

```javascript
Number.isNaN(123); // Returns false Number.isNaN("Hello"); // Returns false
Number.isNaN(NaN); // Returns true
```

In summary, the `isNaN` function is used to check if a value is NaN or not, but it's important to be aware of its behavior with non-numeric values and consider using `Number.isNaN` for more precise NaN checks.

Q3: What is negative Infinity?

Ans. In JavaScript, "Negative Infinity" is a special value representing negative infinity, which is a concept in mathematics and computer science to denote a value that is smaller (more negative) than any real number. It is often used to represent situations where a numerical result is unbounded and tends toward negative infinity.

In JavaScript, you can access negative infinity using the `Number.NEGATIVE_INFINITY` property. For example:

javascriptCopy code
```javascript
var negativeInfinity = Number.NEGATIVE_INFINITY; console.log(negativeInfinity); // Outputs: -Infinity
```

Negative Infinity is typically used in various mathematical and computational contexts. For instance, if you perform a mathematical operation that results in a value that tends toward negative infinity (e.g., dividing a negative number by zero), JavaScript will return negative infinity.

Here's an example:

javascriptCopy code
```javascript
var result = -5 / 0; console.log(result); // Outputs: -Infinity
```

Negative Infinity is used to indicate that a value is too small or too negative to be represented as a finite number. It can be compared to other values, and arithmetic operations with it follow specific rules. For instance, multiplying Negative Infinity by a positive number remains Negative Infinity:

javascriptCopy code
```javascript
var multipliedValue = negativeInfinity * 10; console.log(multipliedValue); // Outputs: -Infinity
```

Negative Infinity is often used in error handling or to signify unbounded behavior in algorithms and calculations. It's one of the special values in JavaScript to represent extreme numerical conditions.

Q4: Which company developed JavaScript?

Ans. JavaScript was developed by Netscape Communications Corporation, a company that was a major player in the early days of the web. The language was created by Brendan Eich in 1995 while he was working at Netscape. Initially, it was called "LiveScript" but was later renamed "JavaScript" to take advantage of the popularity of the Java programming language at the time.

It's important to note that despite the name similarity, JavaScript and Java are different programming languages with distinct purposes and features. JavaScript was designed for client-side web development, enabling web pages to become more interactive, while Java is a separate, general-purpose programming language used in a wide range of applications.

Q5: What are undeclared and undefined variables?

Ans. "Undeclared" and "undefined" are terms used in the context of variables in programming, including JavaScript. They refer to different states of variables:

1. **Undeclared Variables:**
   An undeclared variable is a variable that has been used in code without being declared using a `var`, `let`, or `const` statement. This typically results in a runtime error in JavaScript. Attempting to access or modify an undeclared variable can lead to unpredictable behavior. It's generally considered a best practice to always declare variables before using them to avoid issues with undeclared variables.
   Here's an example of an undeclared variable:

```
x = 10;  // This is an undeclared variable.

console.log(x);
```

1. In this case, `x` is used without being declared, and it will throw a ReferenceError.
2. **Undefined Variables:**
   An undefined variable is one that has been declared but has not been assigned a value. In JavaScript, when you declare a variable without initializing it, the variable is automatically assigned the special value `undefined`. This is a valid state for a variable.
   Here's an example of an undefined variable:

```
var y;

console.log(y);  // This will log "undefined" to the console.
```

1. In this case, `y` has been declared but hasn't been given a value, so it holds the value `undefined`.

It's important to distinguish between these two concepts. Undeclared variables are generally considered errors and should be avoided, while undefined variables are a legitimate state for declared variables. It's good practice to always initialize variables when they are declared to make the code more predictable and less error-prone.

Q6: Write the code for adding new elements dynamically?

Ans. To add new elements dynamically to a web page using JavaScript, you can use the DOM (Document Object Model) manipulation. Here's an example of how to add a new HTML element (in this case, a new paragraph `<p>` element) to the page dynamically:

HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Dynamic Element Addition</title>
</head>
<body>
  <button id="addButton">Add Element</button>
  <div id="container"></div>
</body>
</html>
```

Java Script

```
// Get a reference to the "Add Element" button and the container where we'll add elements.
const addButton = document.getElementById("addButton");
const container =
document.getElementById("container");

// Function to add a new paragraph element when
the button is clicked.
function addNewElement() {
  // Create a new paragraph element.
  const newParagraph =
document.createElement("p");

  // Set the text content for the paragraph.
  newParagraph.textContent = "This is a new
paragraph added dynamically.";

  // Append the new paragraph to the container.
  container.appendChild(newParagraph);
}

// Add a click event listener to the button.
```

```
addButton.addEventListener("click",
addNewElement); // Get a reference to the "Add
Element" button and the container where we'll
add elements.
const addButton =
document.getElementById("addButton");
const container =
document.getElementById("container");

// Function to add a new paragraph element when
the button is clicked.
function addNewElement() {
  // Create a new paragraph element.
  const newParagraph =
document.createElement("p");

  // Set the text content for the paragraph.
  newParagraph.textContent = "This is a new
paragraph added dynamically.";
```

```
  // Append the new paragraph to the container.
  container.appendChild(newParagraph);
}


// Add a click event listener to the button.
addButton.addEventListener("click",
addNewElement);
```

In this example, when the "Add Element" button is clicked, the `addNewElement` function is called. This function creates a new paragraph element, sets its content, and appends it to the `container` div. You can modify this code to create and add different types of elements and customize their content and attributes as needed.

Q7: What is the difference between ViewState and SessionState?

Ans. `ViewState` and `SessionState` are two different concepts used in web development, particularly in ASP.NET, to manage and persist data on web pages. They serve different purposes and have distinct characteristics:

1. **ViewState:**
    - `ViewState` is a mechanism in ASP.NET for maintaining the state of web controls and their values across postbacks. Postbacks occur when a page is submitted to the server and then returned to the client.
    - `ViewState` is used to store data specific to a single web page and its controls. It's primarily used for maintaining the state of controls between postbacks, so that their values remain consistent.
    - Data stored in `ViewState` is kept on the client side, usually in a hidden field on the web page. This data is encoded and sent back to the server with each postback, allowing the server to reconstruct the state of the page.
    - `ViewState` is page-specific, meaning it only retains data for a single page, and the data is typically lost when the user navigates away from the page.
2. **SessionState:**

- `SessionState` is a mechanism for storing and managing user-specific data across multiple pages and multiple HTTP requests during a user's session.
- It allows you to store data on the server side, and this data is associated with a user's session. A session is a period of interaction between a user and a web application, typically starting when a user logs in and ending when they log out or their session times out.
- `SessionState` is used to store information that needs to be available and shared across multiple pages and requests within a user's session. It's commonly used for storing user authentication data, shopping cart contents, and other user-specific information.
- Data stored in `SessionState` is accessible to all pages within a session as long as the user remains logged in, and it is typically retained for a specified period or until the user logs out.

In summary, `ViewState` is used for maintaining the state of controls on a single web page across postbacks, while `SessionState` is used for storing and managing user-specific data across multiple pages and requests during a user's session. They serve different purposes and are used in different contexts within web applications.

Q8: What is === operator?

Ans. The `===` operator, often referred to as the "strict equality operator" or "triple equals," is a comparison operator in JavaScript. It is used to compare two values for equality while considering both their values and their data types. The `===` operator returns `true` if the values being compared are equal in both value and data type; otherwise, it returns `false.

Here's how the `===` operator works:

- If the two values have the same data type and the same value, `===` returns `true`.
- If the two values have different data types or different values, `===` returns `false`.

For example:

5 === 5 // true, because both values are of type number and have the same value

"hello" === "hello" // true, because both values are of type string and have the same value

5 === "5"   // false, because one value is a number, and the other is a string

true === 1   // false, because one value is a boolean, and the other is a number

In contrast to the `===` operator, the loose equality operator `==` (double equals) attempts to perform type coercion, which means it tries to convert the operands to the same data type before comparing them. This can lead to unexpected results in some cases. For this reason, it's often recommended to use the `===` operator (strict equality) to ensure both the value and data type are considered when making comparisons, unless you specifically want to perform type coercion.

Q9: How can the style/class of an element be changed?

Ans. You can change the style and class of an HTML element using JavaScript. Here's how you can do it:

Changing the Style of an Element:

You can modify the inline styles of an HTML element by directly manipulating its `style` property. This property allows you to change individual CSS properties.

// Get a reference to the element you want to style

const element = document.getElementById("myElement");

// Change the background color of the element

element.style.backgroundColor = "blue";

// Change the font size of the element

element.style.fontSize = "18px";

// You can set other CSS properties in a similar way

Alternatively, you can define a CSS class with the desired styles and then add or remove the class from the element using the `classList` property:

```
// Add a CSS class to the element
element.classList.add("my-class");


// Remove a CSS class from the element
element.classList.remove("my-class");


// Toggle a CSS class (add if it doesn't exist, remove if it does)
element.classList.toggle("my-class");
```

Changing the Class of an Element:

You can change the class attribute of an element to apply a different CSS class to it. This is a common way to change the styling of an element, as you can define different styles in your CSS classes.

```
// Get a reference to the element you want to modify
const element = document.getElementById("myElement");


// Change the class of the element to apply a different style
element.className = "new-class";
```

In modern JavaScript, you can also use the `classList` property to add, remove, or toggle classes, as shown in the example above.

Keep in mind that you should have predefined CSS styles or classes in your CSS stylesheet or in a `<style>` tag in your HTML for this approach to work effectively. Changing the class or style dynamically using JavaScript can help create dynamic and interactive web pages.

Q10: How to read and write a file using JavaScript?

Ans. In a web browser environment, JavaScript alone doesn't have direct access to the file system for security reasons. However, you can work with files in a few different ways depending on the context:

1. **Client-Side File Operations (for web applications):**
   If you're working with files in a web application, you can use the File API and HTML5 features to interact with user-selected files. You can read files selected by the user through an `<input type="file">` element and use the FileReader API to read the file's contents. Here's an example of reading a file selected by the user:

```
<input type="file" id="fileInput">

<pre id="fileContent"></pre>
```

JavaScript:

```
const fileInput = document.getElementById("fileInput");

const fileContent = document.getElementById("fileContent");


fileInput.addEventListener("change", function(event) {
  const selectedFile = event.target.files[0];


  if (selectedFile) {
    const reader = new FileReader();


    reader.onload = function(e) {
      fileContent.textContent = e.target.result;
    };
    reader.readAsText(selectedFile);
  }
});
```

1. For writing files in a web application, you can create downloadable files using the Blob and URL.createObjectURL methods, but this doesn't actually write to the user's file system. It allows you to generate files that users can download.
2. **Server-Side File Operations (with Node.js):**

   If you're working with JavaScript on the server side using Node.js, you have more direct access to the file system. You can read and write files using Node.js's built-in `fs` module. Here's an example of reading and writing a file in a Node.js environment:

```
const fs = require("fs");

// Reading a file
fs.readFile("myFile.txt", "utf8", (err, data) => {
  if (err) {
    console.error(err);
  } else {
    console.log(data); // File content
  }
});

// Writing a file
fs.writeFile("newFile.txt", "Hello, World!", (err) => {
  if (err) {
    console.error(err);
  } else {
    console.log("File written successfully.");
  }
});
```

1. In a Node.js environment, you have more control over file operations and can read from and write to the file system.

Remember to handle errors and permissions appropriately when working with files, and be cautious about potential security risks, especially when handling file uploads in web applications.

Q11: What are all the looping structures in JavaScript?

JavaScript provides several looping structures that allow you to repeatedly execute a block of code as long as a certain condition is met. The common looping structures in JavaScript are:

1. **for Loop:** The `for` loop is a versatile loop that is commonly used when you know the number of iterations you want to perform.

```
for (initialization; condition; iteration) {
  // Code to be executed
}
```

Example:

```
for (let i = 0; i < 5; i++) {
  console.log(i); // Prints numbers 0 through 4
}
```

while Loop: The `while` loop continues to execute a block of code as long as a specified condition is true.

```
while (condition) {
  // Code to be executed
}
```

Example:

```
let count = 0;
while (count < 5) {
  console.log(count); // Prints numbers 0 through 4
  count++;
}
```

do...while Loop: The `do...while` loop is similar to the `while` loop, but it guarantees that the code block is executed at least once before checking the condition.

```
do {
  // Code to be executed
} while (condition);
```

Exp.

```
let count = 0;
do {
  console.log(count); // Prints numbers 0 through 4
  count++;
} while (count < 5);
```

for...in Loop: The `for...in` loop is used to iterate over the properties of an object. It's typically used with objects and arrays.

```
for (variable in object) {
  // Code to be executed
}
```

Exp.

```
const person = { name: "John", age: 30 };
```

```
for (let key in person) {
  console.log(key, person[key]); // Prints the property names and values
}
```

for...of Loop: The `for...of` loop is used to iterate over the values of an iterable object, such as arrays, strings, and other collections.

```
for (variable of iterable) {
  // Code to be executed
}
```

Exp.

```
const fruits = ["apple", "banana", "cherry"];
for (let fruit of fruits) {
  console.log(fruit); // Prints each fruit in the array
}
```

forEach Method: The `forEach` method is a built-in method for arrays, and it allows you to iterate through the elements of an array without explicitly defining a loop.

```
array.forEach(function(element, index, array) {
  // Code to be executed
});
```

Exp.

```
const numbers = [1, 2, 3, 4, 5];
numbers.forEach(function(number) {
  console.log(number); // Prints each number in the array
}
```

These are the main looping structures in JavaScript. You can choose the loop that best fits your specific use case based on the problem you're trying to solve and the data you're working with.

Q12: How can you convert the string of any base to an integer in JavaScript?

Ans. You can convert a string representing a number in a different base (e.g., binary, octal, hexadecimal) to an integer in JavaScript using the `parseInt` function. The `parseInt` function allows you to specify the base of the number in the string as the second argument. Here's how you can do it:

```
const binaryString = "1010"; // Binary representation of 10

const decimalNumber = parseInt(binaryString, 2); // Parse as base 2 (binary)


console.log(decimalNumber); // Output: 10
```

In the example above, we have a binary string "1010," and we use `parseInt` with the second argument set to `2` to indicate that it's a binary number. The function then parses the string and converts it to a decimal (base 10) integer, which is assigned to `decimalNumber`.

You can apply the same approach to convert strings in other bases, such as octal or hexadecimal, by changing the base argument accordingly. For example, to convert an octal string to an integer, you can use `parseInt(string, 8)` with a base of `8`. Similarly, for hexadecimal, you can use `parseInt(string, 16)` with a base of `16`.

Here are a few more examples:

```
const octalString = "12"; // Octal representation of 10

const decimalFromOctal = parseInt(octalString, 8);

console.log(decimalFromOctal); // Output: 10


const hexString = "1A"; // Hexadecimal representation of 26
const decimalFromHex = parseInt(hexString, 16);
console.log(decimalFromHex); // Output: 26
```

Keep in mind that it's important to provide the correct base when using `parseInt` to ensure accurate conversion. If the base is not specified or is specified incorrectly, it may lead to unexpected results.

Q12: What is the function of the delete operator?

Ans. The `delete` operator in JavaScript is used to remove a property from an object. Its primary function is to delete a specific property of an object or an element from an array. It can also be used to delete a variable, but this usage is less common and generally discouraged.

The syntax for using the `delete` operator is as follows:

```
delete object.property; // Delete a property from an object
delete array[index];   // Delete an element from an array
delete variable;       // Delete a variable (less common and not recommended)
```

Here are some examples of using the `delete` operator:

Deleting a property from an object:

```
const person = { name: "Alice", age: 30 };
delete person.age; // Deletes the "age" property from the "person" object
console.log(person); // Output: { name: "Alice" }
```

Deleting an element from an array:

```
const fruits = ["apple", "banana", "cherry"];
delete fruits[1]; // Deletes the element at index 1 (i.e., "banana")
console.log(fruits); // Output: ["apple", empty, "cherry"]
```

It's important to note a few things about the `delete` operator:

1. It returns `true` if the deletion is successful, and `false` if it fails, for example, if you try to delete a property that doesn't exist.
2. When you delete an element from an array using `delete`, it leaves an empty slot with `undefined` in place of the deleted element. This doesn't change the array's length.
3. The `delete` operator is primarily used for removing properties from objects. However, it's less commonly used for deleting variables. Deleting variables can lead to unexpected results and is not recommended. Instead, use `var`, `let`, or `const` to control the scope and lifetime of variables.

In many cases, it's better to set a property or element to `null` or `undefined` to effectively "clear" it rather than using `delete`. The use of `delete` is generally more relevant when you need to remove properties from objects dynamically or in specific scenarios, such as working with objects that are tightly controlled, like JavaScript objects representing DOM elements.

Q13: What are all the types of Pop up boxes available in JavaScript?

Ans. **Alert Box:**

- The `alert()` function is used to display a simple pop-up message box with a specified message and an "OK" button.
- It is often used to convey information or provide a message to the user.

Exp.

alert("This is an alert box!");

Confirm Box:

- The `confirm()` function displays a pop-up box with a message and two buttons, "OK" and "Cancel."
- It is commonly used to ask the user for confirmation before proceeding with an action.
- The function returns `true` if the user clicks "OK" and `false` if the user clicks "Cancel."

Exp.

```
const result = confirm("Do you want to proceed?");

if (result === true) {

  // User clicked "OK"

} else {
  // User clicked "Cancel"
}
```

Prompt Box:

- The `prompt()` function displays a pop-up box with a message, an input field for the user to enter text, and "OK" and "Cancel" buttons.
- It is often used to collect input or data from the user.
- The function returns the text entered by the user if they click "OK" or `null` if they click "Cancel."

```
const userInput = prompt("Please enter your name:");
if (userInput !== null) {
  // User entered text
  console.log("Hello, " + userInput);
} else {
  // User clicked "Cancel"
  console.log("No name entered.");
}
```

Q14: What is the use of Void (0)?

Ans. In JavaScript, `void(0)` or simply `void 0` is often used as a special expression to evaluate to `undefined`. It is typically used in the context of generating "void" or "noop" links in HTML, or for preventing a page from navigating away when clicking a hyperlink. Here's how it works:

Void Operator: The `void` operator in JavaScript takes an expression and evaluates it to `undefined`. It's often used with the expression `void(0)` to explicitly return `undefined`. For example:

```
void(0); // Evaluates to undefined
```

Creating "Void" Links: In HTML, when you set the `href` attribute of an anchor `<a>` tag to `javascript:void(0);`, it creates a link that doesn't perform any action when clicked. It's commonly used when you want to make a link inactive or as a placeholder for a link that will be defined later via JavaScript.
Example:

```
<a href="javascript:void(0);">Click me</a>
```

When the user clicks the link, it does nothing and doesn't navigate to a new page, thanks to the `javascript:void(0);` expression.

Preventing Default Navigation: In JavaScript event handling, you may see `void(0)` used to prevent a hyperlink from navigating to a new page when an event handler is triggered, like a click event. This is often done by attaching an event listener to a link and using `void(0)` in the event handler function.
Example:

```
<a href="#" id="myLink">Click me</a>


<script>
  const myLink = document.getElementById("myLink");
  myLink.addEventListener("click", function(event) {
    // Perform some actions
    alert("Link clicked!");


    // Prevent the link from navigating to a new page
    event.preventDefault();
  });
</script>
```

1. In this case, `event.preventDefault()` prevents the default behavior of the link, which is to navigate to the top of the page. You can also use `javascript:void(0)` as an alternative to `event.preventDefault()` in some situations.

It's important to note that while `void(0)` can be used for these purposes, there are more modern and user-friendly ways to achieve the same outcomes in web development. For example, using CSS to style inactive links or using JavaScript to control link behavior and prevent navigation is more common in contemporary web development practices.

Q15:• How can a page be forced to load another page in JavaScript?

Ans. In JavaScript, you can force a web page to load another page by changing the `window.location` object's properties to the desired URL. The most common way to do this is by setting the `window.location.href` property to the new URL. Here's how you can force a page to load another page using JavaScript:

// Change the current page's location to a new URL

window.location.href = "https://www.example.com/new-page.html";

When you set `window.location.href` to a new URL, the current web page will immediately navigate to the specified URL. This effectively forces a page to load a different one. You can use this technique for various purposes, such as redirecting users to a different page, handling form submissions, or implementing client-side routing in a single-page application (SPA).

It's essential to provide a valid URL, including the protocol (e.g., "https://") if it's an external page. If you want to navigate to a different page within the same website, you can use either a relative URL or an absolute URL that doesn't include the protocol and domain.

// Relative URL (navigate within the same website)

window.location.href = "/new-page.html";

// Absolute URL (navigate within the same website)

window.location.href = "https://www.example.com/new-page.html";

Additionally, you can use other `window.location` properties like `window.location.assign()` or `window.location.replace()` for navigation purposes. These methods have slightly different behaviors:

- `window.location.assign("new-url")` loads the new URL, adds a new entry to the browser's history, and allows users to navigate back to the previous page.
- `window.location.replace("new-url")` loads the new URL but replaces the current page in the browser's history. This means users can't navigate back to the previous page using the browser's back button.

What are the disadvantages of using innerHTML in JavaScript?

ANS. While the `innerHTML` property in JavaScript is a convenient way to manipulate the content of HTML elements, it comes with some disadvantages and potential risks, especially when used in certain situations. Here are some of the disadvantages and concerns associated with using `innerHTML`:

1. **Security Risks - Cross-Site Scripting (XSS):** Using `innerHTML` to insert content directly into the DOM without proper sanitization can lead to security vulnerabilities, such as Cross-Site Scripting (XSS) attacks. If user-generated or untrusted content is inserted into the DOM using `innerHTML` without proper validation and encoding, it can execute malicious scripts, potentially compromising the security of your web application.
2. **Performance Overhead:** Manipulating the `innerHTML` of an element can be less efficient than other methods, especially when dealing with large or complex documents. When you modify the `innerHTML` property, the browser has to reparse and re-render the entire element and its descendants, which can be computationally expensive.
3. **Loss of Event Listeners:** If you set the `innerHTML` of an element, any event listeners that were attached to child elements within that element will be lost. You'll need to reattach event listeners after modifying the `innerHTML`.
4. **Potential for Memory Leaks:** Repeatedly modifying the `innerHTML` of an element without properly cleaning up event handlers, references, and objects can lead to memory leaks in some cases. It's essential to be cautious when using `innerHTML` in long-lived applications.
5. **Compatibility and Browser Quirks:** Browser implementations of `innerHTML` can vary, and there might be subtle differences in behavior between browsers. This can lead to cross-browser compatibility issues, especially when dealing with complex HTML structures.
6. **Non-Pure Content Manipulation:** When you use `innerHTML`, you're essentially overwriting the content of an element with a new HTML structure. This approach is less "pure" in the sense that it doesn't directly manipulate the DOM, making it harder to maintain and understand in more complex applications.
7. **Limited Control Over Individual Elements:** When you use `innerHTML`, you typically replace the entire content of an element. If you need to make more granular changes to the DOM, you might find it more challenging to target and manipulate specific elements within the container.

In many cases, it's advisable to use more controlled methods for manipulating the DOM, such as the DOM API methods like `createElement`, `appendChild`, `removeChild`, and `textContent`. These methods allow for more precise control over the DOM and can help mitigate the security risks and performance issues associated with `innerHTML`.

Additionally, consider using libraries and frameworks that offer built-in security mechanisms and abstractions for working with the DOM to help address these concerns.

PRACTICAL

Calculate

```
<!DOCTYPE html>
<html lang="en">


<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-
scale=1.0">
   <title>Document</title>
   <style>
      .section {
         width: 100%;
         height: 100vh;
         background-color: rgba(0, 0, 0, 0.463);
         display: flex;
         justify-content: center;
         align-items: center;
      }


      .calutator {
         width: 400px;
         border: 2px solid black;
         text-align: center;
      }

      input {
         margin: 20px 0;
      }

      .flex {
         display: flex;
         justify-content: space-between;
         align-items: center;
         width: 70%;
         margin: 0 auto;
      }
      button{
         font-size: 20px;
         padding: 3px 5px;
      }
   </style>
</head>

<body>
```

```html
<div class="section">
    <div class="calutator">
        <h3>Maths opration</h3>
        <div class="inp-one">
            <label for="inpOne">Enter 1st number:</label>
            <input type="number" id="inpOne">
        </div>
        <div class="inp-one">
            <label for="inpTwo">Enter 1st number:</label>
            <input type="number" id="inpTwo">
        </div>

        <div class="flex">
            <div class="button">
                <button value="+">+</button>
                <button value="-">-</button>
                <button value="*">*</button>
                <button value="/">/</button>
                <button value="clean">clean</button>
            </div>
            <div class="result">
                <h3 >anser is <span id="answer">00</span></h3>
            </div>
        </div>
    </div>
</div>
<script>
     let anser
    let button =document.querySelectorAll("button")
    let value =document.querySelector("#answer")
    button.forEach(function(e){
        e.addEventListener("click",function(ee){
          let inpOne =  document.getElementById("inpOne").value
          let inpTwo = document.getElementById("inpTwo").value
           if(ee.target.value=="-"){
              anser=parseInt(inpOne)-parseInt(inpTwo)
           }
           if(ee.target.value=="*"){
              anser=parseInt(inpOne)*parseInt(inpTwo)
           }
           if(ee.target.value=="/"){
              anser=parseInt(inpOne)/parseInt(inpTwo)
           }
           if(ee.target.value=="+"){
              anser=parseInt(inpOne)+parseInt(inpTwo)
           }
           if(ee.target.value=="clean"){
              value.innerHTML=`00`
```

```
            }
            value.innerHTML= anser
        })
      })
   </script>
</body>


</html>
```

Color-change

```
<!DOCTYPE HTML>
<HTML LANG="EN">


<HEAD>
   <META CHARSET="UTF-8">
   <META NAME="VIEWPORT" CONTENT="WIDTH=DEVICE-WIDTH,
INITIAL-SCALE=1.0">
   <TITLE>DOCUMENT</TITLE>
</HEAD>


<BODY>
   <DIV STYLE="MARGIN:20PX AUTO; WIDTH: 400PX; TEXT-ALIGN:
CENTER; BOX-SHADOW: 0PX 0PX 6PX BLACK; PADDING: 20PX;"
      CLASS="CONTAINER">
      <DIV CLASS="HEADING">
      <H1>
         CHANGE GACKGROUNDCOLOR
      </H1>
   </DIV>
   <DIV CLASS="BOX">
      <BUTTON CLASS="PLAY">PLAY</BUTTON>
      <BUTTON CLASS="STOP">STOP</BUTTON>
   </DIV>
</DIV>
<SCRIPT>
   // ***********************
   FUNCTION RANDOMCOLOR() {
      LET HAX = "0123456789ABCDEF";
      LET COLOR = "#"
      FOR (I = 0; I < 6; I++) {
         COLOR += HAX[MATH.FLOOR(MATH.RANDOM() * 16)];
      }
      RETURN COLOR;
   }
   // **********************



   LET INTERWAL;
    // **********************
```

```
        FUNCTION PLAY() {
            INTERWAL = SETINTERVAL(BGCHANG, 1000)
            FUNCTION BGCHANG() {

DOCUMENT.QUERYSELECTOR("BODY").STYLE.BACKGROUNDCOLOR =
RANDOMCOLOR();
            }
        }
          // ***********************
        FUNCTION STOPP() {
            CLEARINTERVAL(INTERWAL)
        }
          // ***********************

DOCUMENT.QUERYSELECTOR(".PLAY").ADDEVENTLISTENER("CLICK",
PLAY)

DOCUMENT.QUERYSELECTOR(".STOP").ADDEVENTLISTENER("CLICK",
STOPP)


    </SCRIPT>
</BODY>


</HTML>
```

MARKSHEET

```
<!DOCTYPE HTML>
<HTML LANG="EN">

<HEAD>
  <META CHARSET="UTF-8">
  <META NAME="VIEWPORT" CONTENT="WIDTH=DEVICE-WIDTH,
INITIAL-SCALE=1.0">
  <TITLE>DOCUMENT</TITLE>
  <STYLE>
    .SECTION {
      WIDTH: 100%;
      HEIGHT: 100VH;
      BACKGROUND-COLOR: RGBA(0, 0, 0, 0.635);
      DISPLAY: FLEX;
      JUSTIFY-CONTENT: CENTER;
      ALIGN-ITEMS: CENTER;
    }

    .MARKSHEET {
      WIDTH: 50%;
      BORDER: 2PX SOLID BLACK;
      PADDING: 20PX;
      TEXT-ALIGN: CENTER;
    }
```

```
}

.FLEX {
    DISPLAY: FLEX;
    JUSTIFY-CONTENT: SPACE-BETWEEN;
}

LABEL {
    FONT-SIZE: 25PX;
    PADDING: 10PX;
}

INPUT {
    FONT-SIZE: 25PX;
    MARGIN: 10PX;
}

.BTN {
    TEXT-ALIGN: CENTER;
}

#BTN {
    FONT-SIZE: 22PX;
    BACKGROUND-COLOR: RGBA(0, 0, 255, 0.685);
}
```

```
        .P-5 {
            PADDING: 0PX 50PX;
        }
    </STYLE>
</HEAD>

<BODY>
    <DIV CLASS="SECTION">
        <DIV CLASS="MARKSHEET">
            <H1>
                MARKSHEET FOR INFORMATION TECHNOLOGY
            </H1>
            <H3>
                ENTER MARKS
            </H3>
            <DIV CLASS="BOX">
                <DIV CLASS="FLEX">
                    <LABEL FOR="INPONE">1.C LANGUGE</LABEL>
                    <INPUT TYPE="NUMBER" ID="INPONE" VALUE="0">
                </DIV>
                <DIV CLASS="FLEX">
                    <LABEL FOR="INPTWO">2.C++ LANGUGE</LABEL>
                    <INPUT TYPE="NUMBER" ID="INPWTO" VALUE="0">
    </DIV>
    <DIV CLASS="FLEX">
      <LABEL FOR="INPTHREE">3.DATABASE</LABEL>
      <INPUT TYPE="NUMBER" ID="INPTHREE" VALUE="0">
    </DIV>
    <DIV CLASS="FLEX">
      <LABEL FOR="INPFOUR">4.HTML</LABEL>
      <INPUT TYPE="NUMBER" ID="INPFOUR" VALUE="0">
    </DIV>
    <DIV CLASS="FLEX">
      <LABEL FOR="INPFIVE">5.CSS</LABEL>
      <INPUT TYPE="NUMBER" ID="INPFIVE" VALUE="0">
    </DIV>
    <DIV CLASS="FLEX">
      <LABEL FOR="INPSIX">6.PHP</LABEL>
      <INPUT TYPE="NUMBER" ID="INPSIX" VALUE="0">
    </DIV>
    <DIV CLASS="FLEX">
      <LABEL FOR="INPSAVEN">7.CORE JAVA</LABEL>
      <INPUT TYPE="NUMBER" ID="INPSAVEN" VALUE="0">
    </DIV>
  </DIV>
  <DIV CLASS="BTN">
    <BUTTON ID="BTN">RESULT</BUTTON>
```

```
        </DIV>
        <DIV CLASS="FLEX P-5">
            <H3>TOTAL IS :<SPAN ID="TOTAL">00</SPAN>/350</H3>
            <H3>PERCENTAGE :<SPAN ID="PER">0%</SPAN></H3>
        </DIV>
    </DIV>
</DIV>
<SCRIPT>

    LET BTN = DOCUMENT.GETELEMENTBYID("BTN")
    LET TOTAL = DOCUMENT.GETELEMENTBYID("TOTAL")
    LET PERE = DOCUMENT.GETELEMENTBYID("PER")
    LET TOTALVALUE =350
    BTN.ADDEVENTLISTENER("CLICK", FUNCTION () {
        LET INPUT = DOCUMENT.QUERYSELECTORALL("INPUT")
        LET VALUE = PARSEINT(INPUT[0].VALUE) +
PARSEINT(INPUT[1].VALUE) + PARSEINT(INPUT[2].VALUE) +
PARSEINT(INPUT[3].VALUE) + PARSEINT(INPUT[4].VALUE) +
PARSEINT(INPUT[5].VALUE) + PARSEINT(INPUT[6].VALUE)
        LET PER =(VALUE/TOTALVALUE)*100
        CONSOLE.LOG(PER)
        TOTAL.INNERHTML =VALUE;
        PERE.INNERHTML = PER+"%";
    })
</SCRIPT>
```

```
</BODY>

</HTML>
```