Assignment

Part :1

Web Designing and Devlopment

MODULE: 6 JAVASCRIPT BASIC & DOM

What is JavaScript?

JavaScript is a high-level, interpreted programming language that is widely used for client-side web development to create interactive effects within a web browser. It is also used for server-side development using Node.js and for creating native mobile apps using technologies such as React Native. JavaScript is an object-oriented, dynamic and flexible language that is easy to learn and is used to add interactivity and other dynamic elements to websites.

What is the use of isNaN function?

The **isNaN** function in JavaScript is used to determine whether a value is NaN (Not-a-Number) or not. It returns **true** if the value passed to it is **NaN**, and **false** otherwise. The **isNaN** function is often used to validate user input, especially when the input is expected to be a number, to avoid errors and unexpected behavior in the code.

For example:

```
console.log(isNaN(123));  // false
console.log(isNaN("Hello"));  // true
```

What is negative Infinity?

In JavaScript, **-Infinity** represents negative infinity, which is a numeric value that represents a quantity that is less than negative infinity. This value is obtained by dividing a negative number by zero. **-Infinity** is a special value in the JavaScript language and is used to represent the result of certain mathematical operations, such as when the result of a calculation is smaller than the smallest representable number.

For example:

```
console.log(1 / -0);  // -Infinity
```

Which company developed JavaScript?

JavaScript was developed by Netscape Communications Corporation in the mid-1990s as a scripting language for web browsers. It was created by Brendan Eich, who was a programmer at Netscape at the time. JavaScript was initially called Mocha and was later changed to Livescript before finally being named JavaScript. JavaScript quickly became popular due to its versatility and ability to add

interactivity and dynamic elements to websites, and it is now an essential technology for front-end web development.

What are undeclared and undefined variables?

In JavaScript, variables must be declared using the **var**, **let**, or **const** keywords before they can be used in the code.

An **undeclared** variable is a variable that has not been declared using one of these keywords and is therefore not recognized by the JavaScript interpreter. Attempting to use an undeclared variable will result in a reference error.

An **undefined** variable, on the other hand, is a variable that has been declared using one of the keywords, but has not been assigned a value. The value of an undefined variable is **undefined**.

For example:

```
// undeclared variable
console.log(x);  // Uncaught ReferenceError: x is not defined
// undefined variable
var y;
console.log(y);  // undefined
```

Write the code for adding new elements dynamically?

There are several ways to dynamically add new elements to a web page using JavaScript, here are a few common ones:

1. Using **innerHTML** property: You can use the **innerHTML** property to dynamically add new elements to an existing HTML element.

```
// select the target element
var target = document.getElementById("target");
// add a new element
target.innerHTML = "<p>This is a new paragraph.</p>";
```

2. Using the **createElement** and **appendChild** methods: You can use the **createElement** method to create a new HTML element, and the **appendChild** method to add it to an existing element.

```
// select the target element
var target = document.getElementById("target");

// create a new element
var newElement = document.createElement("p");
newElement.innerHTML = "This is a new paragraph.";

// add the new element to the target
target.appendChild(newElement);
```

3. Using the **insertAdjacentHTML** method: You can use the **insertAdjacentHTML** method to add new HTML elements to an existing element.

```
// select the target element
var target = document.getElementById("target");
// add a new element
target.insertAdjacentHTML("beforeend", "<p>This is a new paragraph.</p>");
```

What is the difference between ViewState and SessionState?

`ViewState` and `SessionState` are both mechanisms used to store data on the server in a web application. However, they differ in their scope, lifetime, and use case.

`ViewState` is a mechanism used to store the state of a web page between postbacks. It is used to retain values of controls on the page, such as text boxes, drop-down lists, and other user inputs, between postbacks. The data is stored as a hidden field in the HTML of the page and is sent back and forth between the client and server with each request and response. The lifetime of ViewState data is limited to the life of the current page and is lost when the user navigates to another page.

`SessionState`, on the other hand, is a mechanism used to store data that is specific to a particular user session. It is used to store data that needs to persist across multiple page requests and multiple postbacks during a user session. The data is stored on the server and can be accessed from any page in the application. The lifetime of SessionState data is maintained for the duration of the user session, which is typically established when the user logs in and lasts until the user logs out or their session times out.

In summary, ViewState is used to store data specific to a single web page, while SessionState is used to store data specific to a user session that needs to persist across multiple pages.

What is === operator?

The `===` operator in JavaScript is known as the strict equality operator. It compares two values for equality and returns `true` if the values are equal and `false` otherwise. Unlike the equality operator `==`, the strict equality operator `===` performs type coercion. That means it compares values after converting both operands to the same type.

For example:

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

console.log("1" === 1);     // false

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

console.log(null === undefined); // false
```

In general, it's recommended to use the strict equality operator `===` instead of the equality operator `==` in JavaScript, as the `===` operator is more predictable and avoids unexpected results from type coercion

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

In JavaScript, you can change the style or class of an HTML element in several ways:

1. Using the **style** property: You can directly access the **style** property of an element and set its CSS properties.

```
// select the element
var element = document.getElementById("target");

// change the style of the element
element.style.backgroundColor = "red";
element.style.height = "100px";
element.style.width = "100px";
```

2. Using the **className** property: You can access the **className** property of an element and set it to the class name you want to apply. This will change the styles applied to the element using the specified class in your CSS.

```
// select the element
var element = document.getElementById("target");

// change the class of the element
element.className = "new-class";
```

3. Using the **setAttribute** method: You can use the **setAttribute** method to add or modify the **class** attribute of an element.

```
// select the element
var element = document.getElementById("target");

// change the class of the element
element.setAttribute("class", "new-class");
```

These are just a few ways to change the style or class of an element in JavaScript. The method you choose will depend on your specific requirements and the structure of your HTML and CSS.

How to read and write a file using JavaScript?

JavaScript has no direct method to read or write local files, as it is a client-side language and it can only access local files through user interaction, like file selection through an file input element.

However, you can use the HTML5 File API to read a local file using JavaScript. Here's an example

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

<script>

 const input = document.getElementById('fileInput');

 input.addEventListener('change', (event) => {

  const file = event.target.files[0];

  const reader = new FileReader();

  reader.onload = function() {

   const text = reader.result;

   console.log(text);

  };

 reader.readAsText(file);

 });

</script>
```

This example creates an input element with the type file, and when the user selects a file, it reads the contents of the file and logs it to the console.

Note: Writing a file using JavaScript is not possible due to security constraints. To save data to the local file system, you need to use a server-side language like PHP, Python, Ruby, or .NET, or use a server-side technology like Node.js, which allows you to run JavaScript on the server.

What are all the looping structures in JavaScript?

In JavaScript, there are two main looping structures: **for** loops and **while** loops.

1. **for** loop: The **for** loop is used to execute a block of code repeatedly a specific number of times. The syntax for a **for** loop is:

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

Here's an example:

```
for (var i = 0; i < 5; i++) {
  console.log(i);
}
```

This will log the numbers 0 to 4 to the console.

2. **while** loop: The **while** loop is used to execute a block of code repeatedly as long as a specified condition is true. The syntax for a **while** loop is:

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

Here's an example:

```
var i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```

This will log the numbers 0 to 4 to the console.

Additionally, there's also the **do...while** loop, which is similar to the **while** loop, but the code in the loop will be executed at least once, even if the condition is false. The syntax for a **do...while** loop is:

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

These are the main looping structures in JavaScript. Loops are useful for repeating operations, and they are a fundamental part of most programming languages.

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

In JavaScript, you can use the **parseInt** function to convert a string representation of a number in any base to an integer. The **parseInt** function takes two arguments: the string to be converted, and the base of the string representation (optional, defaults to base 10).

Here's an example:

```
const binaryString = "1010";
```

```
const decimal = parseInt(binaryString, 2);
console.log(decimal); // Output: 10
```

In this example, the binary string "1010" is converted to the decimal number 10 using the `parseInt` function with a base of 2. The output is then logged to the console.

Note that the `parseInt` function will stop parsing the string as soon as it encounters a non-numeric character. If the string starts with whitespace, the whitespace characters will be ignored.

If the string cannot be parsed into a valid number, `NaN` (Not-a-Number) will be returned. To check if the result is a valid number, you can use the `isNaN` function.

What is the function of the delete operator?

The `delete` operator is used in JavaScript to remove properties from objects. The `delete` operator takes an object property as its operand and, if the property is deletable, removes the property and returns `true`. If the property is not deletable or doesn't exist, the `delete` operator returns `false`.

Here's an example:

```
let obj = { name: "John", age: 30 };

delete obj.age;
console.log(obj); // Output: { name: "John" }

delete obj.gender;
console.log(obj); // Output: { name: "John" }
```

In this example, the `delete` operator is used to remove the `age` property from the `obj` object, which is possible because it was added to the object dynamically. The `delete` operator also returns `true` to indicate that the property was deleted. Then, the `delete` operator is used to remove the non-existent `gender` property from the `obj` object, which returns `false`.

It's important to note that the `delete` operator only works on properties of objects and does not affect variables or object prototypes. Also, the `delete` operator cannot delete properties that are marked as non-configurable or properties of non-extensible objects.

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

In JavaScript, there are several types of pop-up boxes that can be used to interact with users and receive input:

1. **alert**: The **alert** box is a simple pop-up window that displays a message to the user. The **alert**

What is the use of Void (0)?

The `void(0)` expression is a common technique used in JavaScript to prevent the default behavior of a page element, such as a link. When a link is clicked, it typically causes the browser to navigate to a different page. By using `void(0)` as the value of the `href` attribute, the default behavior of the link is cancelled, and the navigation is prevented.

Here's an example:

```
<a href="javascript:void(0);" onclick="alert('Hello World');">Click Me</a>
```

In this example, the `void(0)` expression is used in the `href` attribute of the link to prevent the default behavior of the link. When the link is clicked, the `onclick` event is triggered and an alert box with the message "Hello World" is displayed.

The `void(0)` expression is equivalent to `undefined` and is used because it's a concise way to return `undefined` as the result of an expression. By returning `undefined`, the expression has no effect on the web page and prevents the default behavior of the page element.

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

In JavaScript, you can use the `window.location` object to force a page to load another page. The `window.location` object has a property called `href` that can be used to set the URL of the page to be loaded.

Here's an example:

```
window.location.href = "https://www.example.com";
```

In this example, the `window.location.href` property is set to the URL "https://www.example.com", which causes the current page to navigate to the specified URL.

You can also use the `window.location.replace` method to load another page, which replaces the current page in the history stack and prevents the user from navigating back to the previous page using the browser's back button.

Here's an example:

```
window.location.replace("https://www.example.com");
```

In this example, the `window.location.replace` method is used to load the URL "https://www.example.com", which replaces the current page in the history stack and prevents the user from navigating back to the previous page.

What are the disadvantages of using innerHTML in JavaScript?

1. Security Concerns: Using `innerHTML` to insert dynamic content from untrusted sources can result in security vulnerabilities such as Cross-Site Scripting (XSS).
2. Performance Issues: Modifying the `innerHTML` property can be slow, especially for larger amounts of content, as the browser must re-render the entire contents of the element.
3. Inconsistent Behavior: Different browsers may have different implementations of the `innerHTML` property, leading to inconsistent behavior and difficulties with cross-browser compatibility.
4. Difficult to maintain code: Modifying the contents of an element using `innerHTML` can result in a cluttered and difficult-to-maintain codebase.
5. Lack of Separation of Concerns: Using `innerHTML` to manipulate the DOM tightly couples the structure of the HTML and the behavior of the JavaScript code, making it harder to maintain and test.