**FRONTEND ASSIGNMENT**

**WEB DESIGNING**

**MODULE: 4 (JavaScript Basic & DOM)**

# What is javascript ?

**Ans.**

JavaScript is a programming language that is used to create interactive and dynamic web pages. It is a client-side scripting language that runs on the user's web browser.

JavaScript is used to add functionality to web pages, such as form validation, dynamic content, and interactive features. JavaScript can also be used to create web applications and games.

It is a versatile language that is used by developers to create a wide range of web-based applications. JavaScript is easy to learn and has a large community of developers who contribute to its development.

It is an essential skill for front-end web developers who want to create dynamic and interactive web applications.

1. **What is the use of isNaN function?**

**Ans.** The isNaN() function is used to determine whether a value is not a number. It returns true if the value is not a number, and false if the value is a number. This function is often used in JavaScript to validate user input, especially when dealing with form data.

1. **What is negative Infinity?**

**Ans.**  Negative infinity is a mathematical concept that represents a value that is infinitely small and negative. It is often used to represent the smallest possible value in a given system or set of numbers. In JavaScript, negative infinity is represented by the constant value "-Infinity".

Example:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=”UTF-8”>  <meta Http-equiv=”X-UA-Compatible” content=”IE=edge”>  <meta name=”viewport” content=”width-device-width, initial-scale=1.0”>  <title>document</title>  </head>    <body>      <style>          h1 {              color: green;          }      </style>        <h1>Tops Technology</h1>      <h1>        What is negative infinity in JavaScript?    </h1>        <button onclick="TopsNegativeInfinity()">        Generate negative infinite    </button>        <p id="Tops"></p>        <script>          function TopsNegativeInfinity() {              //negative value greater than the              //largest representable number in JavaScript              var n = (-Number.MAX\_VALUE) \* 2;              document.getElementById("Tops").innerHTML = n;          }      </script>    </body>    </html> |

1. **Which company developed JavaScript?**

**Ans.** JavaScript was developed by Netscape Communications Corporation in collaboration with Sun Microsystems in the mid-1990s.

1. **What are undeclared and undefined variables?**

* **Ans.** [**Undefined**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) **variable means a variable has been declared but does not have a value.**
* **Undeclared** **variable means that the variable does not exist in the program at all.**

**Undefined Example**

**var dog;**

**console.log(dog);**

Output:

**Undefined**

**Undeclared Example**

**console.log(cat);**

**ReferenceError: cat is not defined**

1. **Write the code for adding new elements dynamically?**

**Ans.** Here's an example of adding a new element dynamically to an HTML document using JavaScript:

HTML:

<div id="parent-element">

<p>This is the parent element.</p>

</div>

JavaScript:

// Get the parent element

const parentElement = document.getElementById("parent-element");

// Create a new element

const newElement = document.createElement("p");

// Add content to the new element

const newElementText = document.createTextNode("This is a new element!");

newElement.appendChild(newElementText);

// Add the new element to the parent element

parentElement.appendChild(newElement);

In this example, we first select the parent element that we want to add the new element to using the `getElementById` method. Then, we create a new element using the `createElement` method and add content to it using the `createTextNode` method. Finally, we append the new element to the parent element using the `appendChild` method.

1. **What is the difference between ViewState and SessionState?**

## Ans.

|  |  |
| --- | --- |
| ViewState | SessionState |
| Maintained at page level only. | Maintained at session level. |
| View state can only be visible from a single page and not multiple pages. | Session state value availability is across all pages available in a user session. |
| It will retain values in the event of a postback operation occurring. | In session state, user data remains in the server. Data is available to user until the browser is closed or there is session expiration. |
| Information is stored on the client’s end only. | Information is stored on the server. |
| used to allow the persistence of page-instance-specific data. | used for the persistence of user-specific data on the server’s end. |
| ViewState values are lost/cleared when new page is loaded. | SessionState can be cleared by programmer or user or in case of timeouts. |

1. **What is === operator?**

**Ans.** The `===` operator is a strict equality operator in JavaScript. It compares two values for equality without performing any type coercion. This means that the values being compared must be of the same type in order for the comparison to be true.

For example, `5 === 5` would return `true`, because both values are numbers and are equal. However, `5 === "5"` would return `false`, because the first value is a number and the second value is a string.

The strict equality operator is often used in JavaScript to avoid unexpected type coercion. It is considered good practice to use the `===` operator instead of the loose equality operator `==` whenever possible.

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

**Ans.** There are several ways to change the style or class of an element using JavaScript. Here are a few examples:

1. Changing the style of an element using the `style` property:

const element = document.getElementById("my-element");

element.style.backgroundColor = "red";

In this example, we first select the element that we want to change using the `getElementById` method. Then, we access the `style` property of the element and set the `backgroundColor` property to "red".

2. Changing the class of an element using the `className` property:

const element = document.getElementById("my-element");

element.className = "new-class";

In this example, we first select the element that we want to change using the `getElementById` method. Then, we access the `className` property of the element and set it to "new-class".

3. Changing the class of an element using the `classList` property:

const element = document.getElementById("my-element");

element.classList.add("new-class");

In this example, we first select the element that we want to change using the `getElementById` method. Then, we access the `classList` property of the element and use the `add` method to add the "new-class" class to the element.

These are just a few examples of how to change the style or class of an element using JavaScript. There are many other properties and methods that can be used to manipulate the style and class of elements, depending on the specific use case.

1. **How to read and write a file using JavaScript?**

**Ans.** In JavaScript, you can read and write files using the `File` and `FileReader` APIs. Here's an example of how to read a file:

const input = document.getElementById('file-input');

const file = input.files[0];

const reader = new FileReader();

reader.addEventListener('load', (event) => {

const contents = event.target.result;

console.log(contents);

});

reader.readAsText(file);

In this example, we first select the file input element and get the first file that was selected. We then create a new `FileReader` object and add an event listener for the `load` event. When the file is finished loading, the event listener function is called and we can access the contents of the file using the `result` property of the `event.target`.

Here's an example of how to write a file:

const contents = 'Hello, world!';

const filename = 'example.txt';

const blob = new Blob([contents], { type: 'text/plain' });

const url = URL.createObjectURL(blob);

const link = document.createElement('a');

link.download = filename;

link.href = url;

link.click();

In this example, we first create a string containing the contents of the file that we want to write. We then create a new `Blob` object containing the contents and specify the MIME type of the file. We create a URL for the blob using `URL.createObjectURL`, and then create a new link element with the `download` and `href` attributes set to the filename and URL of the blob, respectively. Finally, we programmatically click the link to download the file.

Note that file I/O in JavaScript is typically limited to the client-side environment, such as a web browser. Server-side file I/O is typically handled by a back-end programming language or framework.

1. **What are all the looping structures in JavaScript?**

**Ans.** There are three types of loops in JavaScript: `for`, `while`, and `do...while`.

A `for` loop is used to iterate over a set of values for a specified number of times. Here's an example:

for (let i = 0; i < 10; i++) {

console.log(i);

}

In this example, the loop will iterate 10 times, with the value of `i` ranging from 0 to 9. The `let` keyword is used to declare a new variable `i` and initialize it to 0. The loop will continue as long as `i` is less than 10, and after each iteration, `i` will be incremented by 1.

A `while` loop is used to iterate over a set of values until a specified condition is met. Here's an example:

let i = 0;

while (i < 10) {

console.log(i);

i++;

}

In this example, the loop will iterate until `i` is greater than or equal to 10. The value of `i` is initialized to 0, and after each iteration, `i` is incremented by 1.

A `do...while` loop is similar to a `while` loop, but it guarantees that the loop body will be executed at least once, even if the condition is false. Here's an example:

let i = 0;

do {

console.log(i);

i++;

} while (i < 10);

In this example, the loop will iterate until `i` is greater than or equal to 10, but the loop body will be executed at least once, even if `i` is initially greater than or equal to 10.

These are the three basic types of loops in JavaScript. You can use them to iterate over arrays, strings, and other types of collections, or to perform other types of repetitive tasks.

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

**Ans.** You can convert a string of any base to an integer in JavaScript using the `parseInt()` function. The `parseInt()` function takes two arguments: the string to be parsed, and the base of the number system used in the string. Here's an example:

let num = parseInt("1010", 2);

console.log(num);

In this example, the string `"1010"` is being parsed as a binary number, so the second argument to `parseInt()` is `2`. The resulting integer value is `10`, which is printed to the console.

You can also use `parseInt()` to parse numbers in other bases, such as hexadecimal:

let num = parseInt("FF", 16);

console.log(num);

In this example, the string `"FF"` is being parsed as a hexadecimal number, so the second argument to `parseInt()` is `16`. The resulting integer value is `255`, which is printed to the console.

Note that if the string cannot be parsed as a valid number in the specified base, `parseInt()` will return `NaN` (Not a Number).

1. **What is the function of the delete operator?**

**Ans.** The `delete` operator in JavaScript is used to delete a property from an object, or to delete an element from an array. Here's an example:

let obj = {a: 1, b: 2, c: 3};

delete obj.b;

console.log(obj);

In this example, the `delete` operator is used to delete the property `b` from the object `obj`. The resulting object has only two properties: `a` and `c`, which are printed to the console.

You can also use the `delete` operator to remove an element from an array:

let arr = [1, 2, 3];

delete arr[1];

console.log(arr);

In this example, the `delete` operator is used to remove the element at index `1` from the array `arr`. The resulting array has only two elements: `1` and `3`, which are printed to the console.

Note that the `delete` operator only removes the property or element from the object or array. It does not affect the length of an array, and it does not remove any references to the deleted value from other parts of the code.

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

**Ans.**  There are three types of pop-up boxes in JavaScript: `alert()`, `confirm()`, and `prompt()`.

The `alert()` function displays a pop-up box with a message and an OK button. Here's an example:

alert("Hello, world!");

In this example, the `alert()` function displays a pop-up box with the message "Hello, world!".

The `confirm()` function displays a pop-up box with a message and two buttons: OK and Cancel. Here's an example:

let result = confirm("Are you sure you want to delete this item?");

if (result) {

// delete the item

}

In this example, the `confirm()` function displays a pop-up box with the message "Are you sure you want to delete this item?" and two buttons: OK and Cancel. The result of the function is stored in the `result` variable. If the user clicks OK, the `result` variable will be `true`, and the code inside the `if` statement will execute.

The `prompt()` function displays a pop-up box with a message, an input field, and two buttons: OK and Cancel. Here's an example:

let name = prompt("What is your name?");

console.log("Hello, " + name + "!");

In this example, the `prompt()` function displays a pop-up box with the message "What is your name?" and an input field. The result of the function is stored in the `name` variable. The code then prints a message to the console using the value of the `name` variable.

Note that the `confirm()` and `prompt()` functions return a boolean value and a string value, respectively. You can use these values in your code to make decisions or to perform actions based on user input.

1. **What is the use of Void (0)?**

**Ans.** The `void(0)` expression is used to evaluate an expression and then return `undefined`. It is often used in HTML anchor tags (`<a>`) to prevent the browser from following the link when the user clicks on it.

Here's an example:

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

In this example, the `href` attribute of the anchor tag is set to `javascript:void(0)`. When the user clicks on the link, the browser will execute the JavaScript expression `void(0)`, which evaluates to `undefined`. This prevents the browser from following the link and refreshing the page.

Note that the `void` operator can be used with any expression. For example, you can use `void(1 + 1)` to evaluate the expression `1 + 1` and then return `undefined`. However, `void(0)` is commonly used in this context because it is a short, simple expression that does not have any side effects.

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

**Ans.**  **Step 1:** Create a file named ***index.html***. Add a heading and two buttons to it. One button forcefully loads a page with a live URL and the other button loads a local HTML page. In the *<script>* tag we have two functions, one loads gfg home page, and the second loads a local HTML page using ***window.location*** property.

|  |
| --- |
| <!DOCTYPE html>  <**html** lang="en">    <**head**>      <**meta** charset="UTF-8">      <**meta** http-equiv="X-UA-Compatible"          content="IE=edge">      <**meta** name="viewport" content=          "width=device-width, initial-scale=1.0">  </**head**>    <**body**>      <**h3**>This is the original page</**h3**>      <**br**>        <**button** onclick="force\_load\_gfg()">          Force Load GFG Page      </**button**>      <**br**><**br**>        <**button** onclick="force\_load\_local()">          Force Load Local HTML page      </**button**>        <**script**>          function force\_load\_gfg() {              window.location =                  "<https://www.geeksforgeeks.org/>"          }            function force\_load\_local() {              window.location =                  "F:/gfg/PageRedirect/newPage.html"          }      </**script**>  </**body**>    </**html**> |

1. **What are the disadvantages of using innerHTML in JavaScript?**

**Ans.** Using `innerHTML` in JavaScript can be risky because it can potentially allow attackers to inject malicious code into your web page. If you use `innerHTML` to insert user-generated content into your web page without properly sanitizing it first, you could be opening up your page to cross-site scripting (XSS) attacks.

XSS attacks occur when an attacker is able to inject malicious JavaScript code into a web page, which can then be executed by other users who view the page. This can allow the attacker to steal sensitive information, such as login credentials or credit card numbers, or to perform other malicious actions on behalf of the user.

To prevent XSS attacks, it's important to properly sanitize any user-generated content that you insert into your web page using `innerHTML`. You should also consider using other methods, such as `textContent`, to insert content into your page that does not allow HTML tags or scripts.

Another disadvantage of using `innerHTML` is that it can be slower than other methods for updating the content of a web page. This is because `innerHTML` requires the browser to parse the HTML content and update the DOM, which can be a relatively slow process. In some cases, it may be faster to use other methods, such as creating new DOM elements and appending them to the page.

**Example:**

|  |
| --- |
| <!DOCTYPE html>  <**html**>    <**head**>  <meta charset=”UTF-8”>  <meta Http-equiv=”X-UA-Compatible” content=”IE=edge”>  <meta name=”viewport” content=”width-device-width, initial-scale=1.0”>      <**title**>          Using innerHTML in JavaScript      </**title**>  </**head**>    <**body** style="text-align: center">        <**h1** style="color:green">          GeeksforGeeks      </**h1**>        <**p** id="P">          A computer science          portal for geeks.      </**p**>        <**button** onclick="geek()">          Try it      </**button**>        <**p** id="p"></**p**>        <**script**>          function geek() {              var x = document.getElementById("P")                          .innerHTML;                document.getElementById("p")                          .innerHTML = x;                document.getElementById("p")                          .style.color = "green";          }      </**script**>  </**body**>    </**html**> |