Module :6 JAVASCRIPT BASIC & DOM

1. What is JavaScript?

JavaScript is a scripting language for creating dynamic web page content. It creates elements for improving site visitors' interaction with web pages, such as dropdown menus, animated graphics, and dynamic background colors.

JavaScript is a lightweight programming language that web developers commonly use to create more dynamic interactions when developing web pages, applications, servers, and or even games. Developers generally use JavaScript alongside **HTML** and **CSS** The scripting language works well with CSS in formatting HTML elements. However, JavaScript still maintains user interaction, something that CSS cannot do by itself.
JavaScript's implementations within the web, mobile application, and game development make the scripting language worth learning. You can do so via learning platforms like **BitDegree** or by exploring free JavaScript templates and applications on code hosting platforms like **GitHub**.

2.What is the use of isNaN function?

The JavaScript **isNaN()** Function is used to check whether a given value is an illegal number or not. It returns true if the value is a NaN else returns false. It is different from the Number.isNaN() Method.

Syntax:

```
isNaN( value )
```

Parameter Values: This method accepts a single parameter as mentioned above and described below:

- **value:** It is a required value passed in the isNaN() function.

Return Value: It returns a Boolean value i.e. returns true if the value is NaN else returns false.

Example: In this example, we will check various values for **isNan()** and the output will be in boolean format.

- Javascript

```javascript
console.log(isNaN(12));
console.log(isNaN(0 / 0));
console.log(isNaN(12.3));
console.log(isNaN("Geeks"));
console.log(isNaN("13/12/2020"));
console.log(isNaN(-46));
console.log(isNaN(NaN));
```

Output:

```
false

true

false

true

true

false
```

```
true
```

3.What is negative Infinity?

The **negative infinity** in JavaScript is a constant value that is used to represent a value that is the lowest available. This means that no other number is lesser than this value. It can be generated using a self-made function or by an arithmetic operation.
Note: JavaScript shows the NEGATIVE_INFINITY value as -Infinity.
Negative infinity is different from mathematical infinity in the following ways:

- Negative infinity results in **-0**(different from 0) when divided by any other number.
- When divided by itself or positive infinity, negative infinity return NaN
- Negative infinity, when divided by any positive number (apart from positive infinity) is negative infinity.
- Negative infinity, divided by any negative number (apart from negative infinity) is positive infinity.
- If we multiply negative infinity with NaN, we will get NaN as a result.
- The product of 0 and negative infinity is Nan.
- The product of two negative infinities is always a positive infinity.
- The product of both positive and negative infinity is always negative infinity.

Syntax:
```
Number.NEGATIVE_INFINITY
```

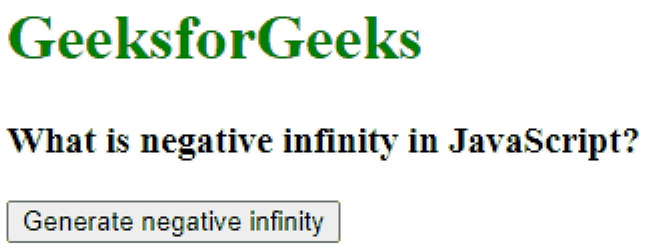
Example 1:

- html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Document</title>
</head>
```

```
<body>
    <h1 style="color: green;">
        GeeksforGeeks
    </h1>
    <h3>
        What is negative infinity in JavaScript?
    </h3>
    <button onclick="geekPositiveInfinity()">
        Generate negative infinity
    </button>
    <p id="geek"></p>
    <script>
        function geekPositiveInfinity() {
            let n = (-Number.MAX_VALUE) * 2;
            document.getElementById("geek").innerHTML =
                "Here the number generated is twice of negative of
Number.MAX_VALUE" + "<br>" +
                " which is lesser than lower limit" + "<br><br>" + n;
        }
    </script>
</body>
</html>
```

Output:



GeeksforGeeks

What is negative infinity in JavaScript?

4. Which company developed JavaScript?

JavaScript was invented by Brendan Eich in 1995. It was developed for Netscape 2, and became the ECMA-262 standard in 1997. After Netscape handed JavaScript over to ECMA, the Mozilla foundation continued to develop JavaScript for the Firefox browser.

5. What are undeclared and undefined variables?

Undefined: It occurs when a variable has been declared but has not been assigned any value. Undefined is not a keyword.
Undeclared: It occurs when we try to access any variable that is not initialized or declared earlier using the *var* or *const keyword*. If we use *'typeof'* operator to get the value of an undeclared variable, we will face the *runtime error* with the return value as **"undefined"**. The scope of the undeclared variables is always global.

6. Write the code for adding new elements dynamically?

```
<!DOCTYPE html>
<html>
<body>

<h1>The Document Object</h1>
<h2>The createElement() Method</h2>
```

```
<p>Create a p element with some text:</p>

<script>
// Create element:
const para = document.createElement("p");
</script>

</body>
</html>
```

Output :

The Document Object

The createElement() Method

Create a p element with some text:

7. What is the difference between ViewState and SessionState?

The basic difference between these two is that the ViewState is to manage state at the client's end, making state management easy for end-user while SessionState manages state at the server's end, making it easy to manage content from this end too. ViewState: It is maintained at only one level that is page-level.

8. What is === operator?

The “===” is a relational operator in some programming languages that allows you to test for physical equality of two references

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Comparison</h1>
<h2>The === Operator</h2>

<p>Assign 5 to x, and display the value of the comparison (x ===
5):</p>

<p id="demo"></p>

<script>
let x = 5;
document.getElementById("demo").innerHTML = (x === 5);
</script>

</body>
</html>
```

Output:

JavaScript Comparison

The === Operator

Assign 5 to x, and display the value of the comparison (x === 5):

true

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

In this article, we will learn how we can change the style/class of an element. If you want to build a cool website or app then UI plays an important role. We can change, add or remove any CSS property from an HTML element on the occurrence of any event with the help of JavaScript. There are two common approaches that allow us to achieve this task.

- style.property
- Changing the class itself

Approach 1: Changing CSS with the help of the style property:

Syntax:

```
document.getElementById("id").style.property = new_style
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>
<p>Changing the HTML style:</p>
```

```
<p id="p1">Hello World!</p>
<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
document.getElementById("p2").style.fontFamily = "Arial";
document.getElementById("p2").style.fontSize = "larger";
</script>


</body>
</html>
```

Output:

JavaScript HTML DOM

Changing the HTML style:

Hello World!

Hello World!

10. How to read and write a file using JavaScript?

On the client side, you can’t read or write files in JavaScript browsers. The fs module in Node.js may be used to accomplish this on the server-side. It has methods for reading and writing files on the file system that are both synchronous and asynchronous. Let’s demonstrate some examples of reading and writing files with the node.js fs module.

The fs.readFile() and rs.writeFile() methods are used to read and write of a file using javascript. The file is read using the fs.readFile() function, which is an inbuilt method. This technique reads the full file into memory and stores it in a buffer.

Syntax:

```
fs.readFile( file_name, encoding, callback_function )
```

Parameters:

- **filename:** It contains the filename to be read, or the whole path if the file is saved elsewhere.
- **encoding:** It stores the file’s encoding. ‘utf8’ is the default setting.
- **callback function:** This is a function that is invoked after the file has been read. It requires two inputs:
- **err:** If there was an error.
- **data:** The file’s content.
- **Return Value:** It returns the contents contained in the file, as well as any errors that may have occurred.

The fs.writeFile() function is used to write data to a file in an asynchronous manner. If the file already exists, it will be replaced.

Syntax:

```
fs.writeFile( file_name, data, options, callback )
```

Parameters:

- **file_name**: It’s a string, a buffer, a URL, or a file description integer that specifies the location of the file to be written. When you use a file descriptor, it will function similarly to the fs. write() method.
- **data**: The data that will be sent to the file is a string, Buffer, TypedArray, or DataView.
- **options:** It’s a string or object that may be used to indicate optional output options. It includes three more parameters that may be selected.
- **encoding**: It’s a string value that indicates the file’s encoding. ‘utf8’ is the default setting.
- **mode**: The file mode is specified by an integer number called mode. 0o666 is the default value.
- **flag**: This is a string that indicates the file-writing flag. ‘w’ is the default value.
- **callback**: This function gets invoked when the method is run.
- **err**: If the process fails, this is the error that will be thrown.

11. What are all the looping structures in JavaScript?

Loops can execute a block of code a number of times.

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

You can write:

```
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

Try it Yourself »

Different Kinds of Loops

JavaScript supports different kinds of loops:

- `for` - loops through a block of code a number of times
- `for/in` - loops through the properties of an object
- `for/of` - loops through the values of an iterable object
- `while` - loops through a block of code while a specified condition is true
- `do/while` - also loops through a block of code while a specified condition is true

The For Loop

The `for` statement creates a loop with 3 optional expressions:

```
for (expression 1; expression 2; expression 3) {
  // code block to be executed
}
```

Expression 1 is executed (one time) before the execution of the code block.

Expression 2 defines the condition for executing the code block.

Expression 3 is executed (every time) after the code block has been executed.

Example

```
for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
```

Try it Yourself »

From the example above, you can read:

Expression 1 sets a variable before the loop starts (let i = 0).

Expression 2 defines the condition for the loop to run (i must be less than 5).

Expression 3 increases a value (i++) each time the code block in the loop has been executed.

ADVERTISEMENT

Expression 1

Normally you will use expression 1 to initialize the variable used in the loop (let i = 0).

This is not always the case. JavaScript doesn't care. Expression 1 is optional.

You can initiate many values in expression 1 (separated by comma):

Example

```
for (let i = 0, len = cars.length, text = ""; i < len; i++) {
  text += cars[i] + "<br>";
}
```

Try it Yourself »

And you can omit expression 1 (like when your values are set before the loop starts):

Example

```
let i = 2;
let len = cars.length;
let text = "";
for (; i < len; i++) {
  text += cars[i] + "<br>";
}
```

Try it Yourself »

Expression 2

Often expression 2 is used to evaluate the condition of the initial variable.

This is not always the case. JavaScript doesn't care. Expression 2 is also optional.

If expression 2 returns true, the loop will start over again. If it returns false, the loop will end.

If you omit expression 2, you must provide a **break** inside the loop. Otherwise the loop will never end. This will crash your browser. Read about breaks in a later chapter of this tutorial.

Expression 3

Often expression 3 increments the value of the initial variable.

This is not always the case. JavaScript doesn't care. Expression 3 is optional.

Expression 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.

Expression 3 can also be omitted (like when you increment your values inside the loop):

Example

```
let i = 0;
let len = cars.length;
let text = "";
for (; i < len; ) {
  text += cars[i] + "<br>";
  i++;
}
```

Try it Yourself »

Loop Scope

Using `var` in a loop:

Example

```
var i = 5;

for (var i = 0; i < 10; i++) {
  // some code
}

// Here i is 10
```

Try it Yourself »

Using `let` in a loop:

Example

```
let i = 5;

for (let i = 0; i < 10; i++) {
  // some code
}

// Here i is 5
```

Try it Yourself »

In the first example, using `var`, the variable declared in the loop redeclares the variable outside the loop.

In the second example, using `let`, the variable declared in the loop does not redeclare the variable outside the loop.

When `let` is used to declare the i variable in a loop, the i variable will only be visible within the loop.

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

In JavaScript **parseInt()** function (or a method) is used to convert the passed-in string parameter or value to an integer value itself. This function returns an **integer** of the base which is specified in the second argument of the **parseInt() function**. JavaScript parseInt() function returns Nan(not a number) when the string doesn't contain a number. We can convert a string to javascript by the following methods:

- Using the parseInt() method
- Using the Number() method
- Using the Unary operator

Using the parseInt() method: JavaScript parseInt() Method is used to accept the string and radix parameter and convert it into an integer.

Syntax:

parseInt(Value, radix)

Example 1: In this example, we will convert a string value to an integer using the **parseInt()** function with no second parameter.**JavaScript ParseInt()** function converts the number which is present in any base to base 10. It parses a string and converts it until it faces a string literal and stops parsing.

- javascript

```
function convertStoI() {
    let a = "100";
    let b = parseInt(a);
    console.log("Integer value is" + b);
    let d = parseInt("3 11 43");
    console.log('Integer value is ' + d);
}
convertStoI();
```

Output:

```
Integer value is 100

Integer value is 3
```

Example 2: In this example, we will convert a string value to an integer using the **parseInt() function** **with a second parameter**.

- javascript

```
function convertStoI() {
    let r = parseInt("1011", 2);
    let k = parseInt("234", 8);
    console.log('Integer value is ' + r);
    console.log("integer value is " + k);
    console.log(parseInt("528GeeksforGeeks"));
}
convertStoI();
```

Output:

```
Integer value is 11

integer value is 156

528
```

Using the Number() method: In Javascript, the **Number()** method is used to convert any primitive data type to a number, if it is not convertible it returns NAN.
Syntax:
Number(value)

Example: In this example, we will see the use of the Number() method.

- Javascript

```
let age = "23";
let name = "Manya";
console.log(Number(age));
console.log(Number(name)) ;
```

Output:
```
23

NaN
```

Using the Unary Operator: In Javascript, the **Unary operator(+)** is used to convert a string, boolean, and non-string to a number.
Syntax:
+op;

Example: In this example, we will see the use of the unary operator.

- Javascript

```
let age = "23";
let name = "Manya";
const number = '100';
console.log(+age);
console.log(+name);
console.log(+number);
```

Output:

```
23

NaN

100
```

13. What is the function of the delete operator?

Delete is comparatively a lesser-known operator in JavaScript. This operator is more specifically used to delete JavaScript object properties.

The JavaScript **pop()**, **shift()**, or **splice()** methods are available to delete an element from an array. But because of the key-value pair in an object, deleting is more complicated. Note that, the delete operator only works on objects and not on variables or functions.

Syntax:

```
delete object

// or

delete object.property

// or

delete object['property']
```

Parameter: It does not take any parameter.

Return type: This operator returns *true* if it removes a property. While deleting an object property that doesn't exist will return a *true* but it will not affect the object. Though while trying to delete a variable or a function will return a *false*.

Below are examples of the delete Operator.

Example:

- javascript

```javascript
let emp = {
    firstName: "Raj",
    lastName: "Kumar",
    salary: 40000
}

console.log(delete emp.salary);
console.log(emp);
```

Output:

```
true

{"firstName":"Raj","lastName":"Kumar"}
```

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

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

15. What is the use of Void (0)?

void is an important keyword in JavaScript which can be used as a unary operator that appears before its single operand, which may be of any type. This operator specifies an expression to be evaluated without returning a value.

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

In JavaScript, we can use window.location object to force a page to load another page. We can use the location object to set the URL of a new page. There are different ways – window.location.href property, window.location.assign() and window.location.replace() methods, to set the URL of a new page using the location object.

```
<html>
<body>
  <h2>Forced Load page using window.location.href property</h2>
  <p>Click on the below button to force reload new page</p>
  <button onclick="forceLoad()">Load</button>
  <script>
    function forceLoad() {
      window.location.href = "https://www.tutorialspoint.com";
    }
  </script>
</body>
 </html>
```

17. What are the disadvantages of using innerHTML in JavaScript?

- **The use of innerHTML very slow:** The process of using innerHTML is much slower as its contents as slowly built, also already parsed contents and elements are also re-parsed which takes time.
- **Preserves event handlers attached to any DOM elements:** The event handlers do not get attached to the new elements created by setting innerHTML automatically. To do so one has to keep track of the event handlers and attach it to new elements manually. This may cause a memory leak on some browsers.
- **Content is replaced everywhere:** Either you add, append, delete or modify contents on a webpage using innerHTML, all contents is replaced, also all the DOM nodes inside that element are reparsed and recreated.
- **Appending to innerHTML is not supported:** Usually, += is used for appending in JavaScript. But on appending to an Html tag using innerHTML, the whole tag is re-parsed.
- **Old content replaced issue:** The old content is replaced even if object.innerHTML = object.innerHTML + 'html' is used instead of object.innerHTML += 'html'. There is no way of appending without reparsing the whole innerHTML. Therefore, working with innerHTML becomes very slow. String concatenation just does not scale when dynamic DOM elements need to be created as the plus' and quote openings and closings becomes difficult to track.

- **Can break the document:** There is no proper validation provided by innerHTML, so any valid HTML code can be used. This may break the document of JavaScript. Even broken HTML can be used, which may lead to unexpected problems.
- **Can also be used for Cross-site Scripting(XSS):** The fact that innerHTML can add text and elements to the webpage, can easily be used by malicious users to manipulate and display undesirable or harmful elements within other HTML element tags. Cross-site Scripting may also lead to loss, leak and change of sensitive information.