**Q-1. What is JavaScript?**

A dynamic computer programming language is called JavaScript. Its implementations enable client-side script to interact with users and create dynamic pages, and it is most frequently used as a component of web pages. It is an object-oriented programming language that may be interpreted. LiveScript was the original name of JavaScript, but Netscape changed it to JavaScript, maybe in response to the buzz that Java was causing. LiveScript, the predecessor to JavaScript, had its debut in Netscape 2.0 in 1995. Netscape, Internet Explorer, and other web browsers all contain the general-purpose language's core.

* A simple, interpreted programming language is JavaScript.
* designed for the development of network-centric applications.
* linked with and complementary to Java.
* connected with and complementary to HTML.
* Open and platform-neutral

**Client-Side JavaScript**

Client-side The most popular variation of the language is JavaScript. For the script's code to be recognized by a browser, it must be incorporated into or referenced from an HTML document.

It implies that a web page need not be static HTML, but rather may contain programs that communicate with users, manage browsers, and generate HTML content on the go.

Over typical CGI server-side scripts, the JavaScript client-side method offers a number of benefits. JavaScript, for instance, can be used to determine whether a user has supplied a valid email address in a form field.

When a user submits a form, JavaScript is run, and only if all of the entries are correct are they sent to the web server.

User-initiated events, including as button clicks, link navigation, and other explicit or implicit user activities, can be caught using JavaScript.

**Advantages of JavaScript**

The merits of using JavaScript are −

* **Less server interaction** and the ability to validate user input before to transmitting the page to the server are two benefits of using JavaScript. Because of the reduced server traffic, your server will be under less stress.
* **The visitors receive immediate** feedback and don't have to wait for the page to reload to find out whether they forgot to type something.
* **Enhanced interactivity** – You may design interfaces that respond when the user moves their mouse cursor over them or presses a keyboard shortcut to activate them.
* **Richer interfaces** You can use JavaScript to integrate sliders and drag-and-drop elements to provide your site visitors with a Rich Interface.

**Limitations of JavaScript**

JavaScript can't be used as a legitimate programming language. It lacks the following crucial components:

* Client-side It is not possible to read or write files using JavaScript. This is still here for security purposes.
* Because there is no such support, JavaScript cannot be utilized for networking applications.
* JavaScript lacks the ability to use multiple processors or threads.

**Q-2. What is the use of isNaN function?**

The isNaN() function in JavaScript can be used to determine whether a given value is or is not an unlawful number. If the value is a NaN, it returns true; otherwise, it returns false. The Number is not the same as it.isNaN() Function.

**Syntax:**

isNaN( value )

**Values for the single parameter** that this method accepts are as follows and are mentioned above:

* **value:** It is a mandatory argument for the isNaN() method.

**The result of this example** isNan() function will be in boolean format and will check various values.

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));  
console.log(isNaN(12));

**Output:**

true  
false  
true  
true  
false  
true  
false

**Q-3. What is negative Infinity?**

In Java script, a negative infinity is a number that is created by "dividing a negative number by zero."

It is not necessary to create a number object in order to access this static attribute.

The negative value of the infinity property of the global object is the same as the value of negative infinity.

**In contrast to the mathematical infinity, the values react differently:**

1. Any positive value multiplied by NEGATIVE\_INFINITY is NEGATIVE\_INFINITY, including POSITIVE\_INFINITY.
2. When a negative value is multiplied by another negative value, the result is POSITIVE\_INFINITY. This includes NEGATIVE\_INFINITY.
3. Zero is NaN when multiplied by NEGATIVE\_INFINITY.
4. NaN is NaN when multiplied by NEGATIVE\_INFINITY.
5. NEGATIVE\_INFINITY is POSITIVE\_INFINITY when divided by any other negative number besides NEGATIVE\_INFINITY.
6. NEGATIVE\_INFINITY is NEGATIVE\_INFINITY when divided by any positive value other than POSITIVE\_INFINITY.
7. Negative\_infinity is NaN when divided by either negative or positive infinite.
8. Zero is the result of any division by NEGATIVE\_INFINITY.

**Q-4. Which company developed JavaScript?**

**Brendan Eich** developed JavaScript while working at **Netscape Communications** in 1995. JavaScript was created by Netscape and Eich as a scripting language for use with Netscape Navigator, the company's primary web browser. In order to portray JavaScript as a companion for the Java language, a product of their partner Sun Microsystems, Netscape changed the language's name from LiveScript to JavaScript. Despite some superficial grammatical similarities, JavaScript and the Java programming language are unrelated.

Following its release, more and more browsers began to support JavaScript. JavaScript was nevertheless not regarded as a serious programming language for a significant portion of its history. Early versions had significant speed and security problems, but there were no other options available to developers. They had to utilize JavaScript if they wanted to run programs in the browser.

**Q-5. What are undeclared and undefined variables?**

When a variable is declared but not given a value, it is said to be undefined. A keyword is not undefined.

When we attempt to access a variable that has not yet been initialized or declared using the var or const keyword, we get the undeclared error. We will get a runtime error with the return value of "undefined" if we attempt to retrieve the value of an undeclared variable using the 'typeof' operator. The undeclared variables always have a global scope.

**For example:** **Undefined:**

undefined  
console.log(geek)  
let geek;

**Undeclared:**

// ReferenceError: myVariable is not define  
console.log(myVariable)

**Example 1:** This example illustrates a situation where an undeclared variable is used.

function GFG() {  
  
// 'use strict' verifies that no undeclared  
// variable is present in our code  
'use strict';  
x = "GeeksForGeeks";  
}  
  
GFG(); // Accessing the above function

**Output**

ReferenceError: x is not defined

**Example 2:** This example checks whether a given variable is undefined or not.

function checkVar() {  
let string;  
  
if (typeof variable === "undefined") {  
string = "Variable is undefined";  
} else {  
string = "Variable is defined";  
}  
  
console.log(string);  
}  
  
checkVar();

**Output**

Variable is undefined

**Q-6. Write the code for adding new elements dynamically?**

**HTML DOM**, which displays an HTML document as a tree-structure, makes HTML documents simple to read and alter. A Document object is created when an HTML file is loaded in a browser window. You can use the **document.createElement()** method to dynamically create a specific HTML element in JavaScript. After creation, attributes can be added. You must put the element into the document's DOM-tree if you want it to appear in the document.

**example**

<html>  
  <body>  
   <button onclick="create()">Create  
Heading</button>  
   <script>  
     function create() {  
     var h1 = document.createElement  
('h1');  
     h1.textContent = "New Heading!!!";  
     h1.setAttribute('class', 'note');  
     document.body.appendChild(h1);  
   }  
   </script>  
  </body>  
</html>

Please record.The element is made when **document.createElement** is used in conjunction with an HTML tag. Using **setAttribute**, the class attribute is changed after the textContent has been adjusted. A data attribute or any other kind of attribute, similar to how you can do so in HTML, might also be added using this. Using the appendChild method of the body element, the element is then finally attached to the body.

It's actually essentially the same as ❮h1 class="note" ❯ New Heading!!! ❮ /h1 ❯.

The ability to create and style pieces "on the fly" is the strength of what we see today. You may either add every element to the page as you develop it or you can use the **createElement()** method to create and insert HTML elements dynamically at runtime.

**Q-7. What is the difference between ViewState and SessionState?**

**Differentiating ViewState from SessionState**

|  |  |
| --- | --- |
| **ViewState** | **SessionState** |
| solely at the page level maintained. | held at the session level. |
| Only one page, not numerous pages, can display the view status. | The availability of session state values spans all pages accessible during a user session. |
| In the case that a postback operation does place, values will be retained. | In session state, user data remains in the server. Data is available to user until the browser is closed or there is session expiration. |
| Only the client's end of the information is kept on file. | On the server, information is saved. |
| used to provide the storage of information unique to a given page instance. | used to store user-specific data permanently on the server. |
| When a new page is loaded, ViewState values are lost or cleared. | It is possible for a programmer, user, or timeout to clear the session state. |

**Q-8 .What is === operator?**

The JavaScript Strict Equality Operator compares two operands and returns true if the operands' values and types match. Since there is no type conversion, the operation will return false even if the value stored in the operands is the same but their types are different.

**Syntax:**

a===b

**Example 1:** In this example, we will compare the value of the same data types.

let a = 2, b=2, c=3;  
let d = {name:"Ram"};  
let e = {name:"Ram"};  
let f = e;  
  
console.log(a===b);  
console.log(a===c);  
console.log(d===e);  
console.log(f===e);

**Output:**When two objects are compared, their references are examined, just like with other comparisons, and true is only returned if the references are identical.

true  
false  
false  
true

**Example 2:** In this example, we will use the strict equality operator on different data types.

let a = 2;  
let b= "2";  
let c = true;  
let d = null;  
let e = undefined;  
  
console.log(a===b);  
console.log(a===c);  
console.log(d===e);

**Output:** Even though some values are the same but still false is returned as the data type is different.

false  
false  
false

**Q - 9 How can the style/class of an element be changed?**

This article will teach us how to modify an element's style or class. The UI is crucial if you want to create a cool website or app. With the use of JavaScript, any CSS property from an HTML element can be modified, added, or removed in response to any event. We are able to complete this work using one of two popular methods.

* style.property
* Changing the class itself

**Approach :**CSS can be modified using the style property to:

**Syntax:**

document.getElementById("id").style.property = new\_style

**Example:**In this instance, a PAN number validator has been constructed. We will first use a regex pattern to match the input value. If it does, add an inline style to the p> tag using JavaScript. If not, change the style of the p> tag.

<!DOCTYPE html>  
<html lang="en">  
  
<body>  
  
<h2>  
How can the style/class of  
an element be changed?  
</h2>  
  
<b>Validate Pan Number</b>  
  
<input type="text" id="pan" />  
<p></p>  
<button id="submit">Validate</button>  
<script>  
  
const btn = document.getElementById("submit");  
btn.addEventListener("click", function () {  
const pan = document.getElementById("pan").value;  
const para = document.querySelector("p");  
let regex = /([A-Z]){5}([0-9]){4}([A-Z]){1}$/;  
if (regex.test(pan.toUpperCase())) {  
para.innerHTML = "Hurrey It's correct";  
  
// Inline style  
para.style.color = "green";  
} else {  
para.innerHTML = "OOps It's wrong!";  
  
// Inline style  
para.style.color = "red";  
}  
});  
</script>  
  
</html>

**Output:**

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

**Validate Pan Number** 

Validate

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

File reading and writing are not supported on the client side by JavaScript browsers. The Node.js fs module can be used for this on the server-side. It has techniques for reading and writing files on the file system that are both synchronous and asynchronous. Several examples of the node.js fs module reading and writing files will be displayed.

To read and write files, utilize Javascript's fs.readFile() and rs.writeFile() APIs. The file is read via an internal technique called fs.readFile(). This approach reads the entire file into memory and stores it in a buffer.

**Syntax:**

fs.readFile( file\_name, encoding, callback\_function )

**Parameters:**

* **filename:** It includes the filename to read or, if the file was saved somewhere else, the entire path.
* The encoding of the file is kept there. The default value is 'utf8'.
* This is a **callback**function that is used after the file has been read. There are two inputs needed.
* **err:** if there was a mistake
* **data:** The contents of the file.
* **Return Value:** It provides both the file's contents and any faults that might have happened.

Data is written asynchronously to files using the fs.writeFile() function. The file will be replaced if it already exists.

**Syntax:**

fs.writeFile( file\_name, data, options, callback )

**Parameters:**

* **<file\_name:< strong="">The location of the file to be written is specified by a string, buffer, URL, or integer file description. A file descriptor will perform similarly to the fs. write() method when used.</file\_name:<>**
* **data: A string, buffer, typed array, or data view are the possible types of data that will be transmitted to the file.**
* **options: This string or object can be used to denote output options that are optional. There are three other criteria available for selection.**
* **The string value encoding describes the file's encoding. The default value is 'utf8'.**
* **mode is an integer number that indicates the file mode. The default value is 0o666.**
* **flag: This string represents the flag for writing files. The default value is "w."**
* **callback: When the method is executed, this function is called.**
* **Err: This is the error that will be thrown if the process fails.**

**Let's use the following example to explain how to write and read files:**

**The writeFile() method is used to create the file in the example below, and the readFile() method is used to read its contents.**

**var fs = require("fs");  
console.log(" Writing into an file ");  
  
// Sample.txt is an empty file  
fs.writeFile(  
"sample.txt",  
"Let's write a few sentences in the file",  
function (err) {  
if (err) {  
return console.error(err);  
}  
  
// If no error the remaining code executes  
console.log(" Finished writing ");  
console.log("Reading the data that's written");  
  
// Reading the file  
fs.readFile("sample.txt", function (err, data) {  
if (err) {  
return console.error(err);  
}  
console.log("Data read : " + data.toString());  
  
});  
}  
);**

**Q-11. What are all the looping structures in JavaScript?**

**In the last chapter, we discovered how to use conditional statements in JavaScript by using practical programs.**

**We learn about looping and jump control statements in this chapter. Code blocks can be run in loops as many times as necessary.**

**In JavaScript, the following looping statements are supported:**

**In Loop**

**During Loop**

**Loop while doing**

**For Loop Statement**

**When a loop is used, it performs the three specified tasks (Statement 1, Statement 2, Statement 3) a fixed number of times. The general syntax of the "for" statement is demonstrated in the following code.**

**Syntax**

**for (initialization; condition; increment/ decrement)  
{  
Executed these statements;  
}**

**While Loop Statement**

**Of all looping statements, the while one is the most straightforward. While a particular condition is true, a block of code is repeated in a while loop. The three distinct jobs have been carried out in the for loop. The same three specific activities are carried out in the while loop as well as where it is mentioned in the statement, but the initialization and conditions of the while loop take place in various locations. The code below demonstrates the while statement's basic syntax:**

**Syntax**

**while (condition)  
{  
Execute these statements; //true  
Increment/decrement;  
}**

**do-while Statement**

**The while statement and the do-while statement are related. The condition is present outside the statement in the do-while construct. If the condition is not met, it will still run at least once. If the condition is true, the code will continue to run; if the condition is false, the code will cease running. The general syntax of the while statement is shown in the following code:**

**Syntax**

**do  
{  
Execute the statements;  
Increment/decrements;  
}  
while (condition) //true**

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

**JavaScript uses the parseInt(), Number(), and Unary operator (+) functions to convert a string to an integer. When the string doesn't contain a number, the parseInt() function returns NaN (not a number). Only the specified number will be returned if a string containing a number is given. Spaces are not permitted in this function. If a specific number is supplied with spaces, the portion of the number that appears before the space is returned as the output.**

**We can use the parseInt(), Number(), and Unary operator (+) to turn a string into an integer. An illustration is used to discuss each of them.**

**Using the parseInt() method**

**The parseInt() method's syntax is**

**parseInt(string); parseInt(string,radix);**

**Where,**

**The string to be parsed is called string.**

**In the Number System, the radix is used to refer to the basis. 8 is octal, 10 is decimal, 16 is hexadecimal, and 2 is binary.**

**The integer value that was extracted from the string is returned by this procedure.**

**Using the Number() method**

**The Number() method's syntax is**

**Number(value)**

**If value is not of a primitive data type that can be converted to a number, NaN is returned. A number is returned by this procedure.**

**Using the unary operator**

**The unary plus operator's syntax is.**

**+op;**

**where an operand is op. The unary operator + tries to turn the operand into a number when it is put before it. The response is a number.**

**Q - 13. What is the function of the delete operator?**

**An object, an object's property, or an element from an array can all be deleted using the delete operator. Variables that are not declared using the var statement can likewise be deleted by the operator.**

**Syntax**

**delete objectName delete objectName.property delete objectName[index] delete property // The command acts only within a with statement.**

**Parameters**

**objectName: An object's name.**

**property: The property already exists.**

**index: An integer that represents the index of the array.**

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

**In general, javascript supports three different forms of pop-up boxes:**

* **Alert: A message with an OK button is all that is shown.**

**< !DOCTYPE html >  
  
< html >  
  
< body >  
  
< ;p >Click the button to display an alert box. < /p >  
  
<button onclick="myFunction()"> Click me </button>  
  
<script>  
function myFunction() {  
alert("Hello! Welcome to edureka!");  
}  
</script>  
  
</body>  
</html>**

* **Confirm -- A confirmation message window with an OK and Cancel button appears.**

**< !DOCTYPE html >  
  
< html >  
  
< body >  
  
< ;p >Click the button to display an alert box. < /p >  
  
<button onclick="myFunction()"> Click me </button>  
  
<script>  
function myFunction() {  
confirm("Press a button!");  
}  
</script>  
  
</body>  
</html>**

* **Prompt: A dialog window asking for the user's input then displays confirmation buttons.**

**< !DOCTYPE html >  
  
< html >  
  
< body >  
  
< ;p >Click the button to display an alert box. < /p >  
  
<button onclick="myFunction()"> Click me </button>  
  
< id ="demo"></p>  
  
<script>  
function myFunction() {  
var person = prompt("Please enter your name", "Edureka");  
if (person != null) {  
document.getElementById("demo").innerHTML =  
"Hello " + person + "! How are you today?";  
}  
}  
  
</script>  
  
</body>  
</html>**

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

**"javascript:void(0)" may have occasionally appeared in an HTML document. It is frequently used when adding an expression to a web page can have unintended consequences. The command "javascript:void(0)" is used to reverse this effect. This expression yields a primitive value that is undefinable. This is frequently applied to hyperlinks. You may occasionally choose to execute some JavaScript from a link. Depending on the URL provided, the browser typically loads a fresh copy of the current page when you click a link. But if you've connected some JavaScript thereto link, you probably don't want this to happen. Using void(0) would stop the page from refreshing.**

**Example:**

**<h1 style="color:green">GeeksforGeeks</h1>**

**<h3>without JavaScript:void(0)</h3>**

**<a href="#" ondblclick="alert('Welcome to Geeks for Geeks')">**

**Double click on me </a>**

**<a href="#" ondblclick="geeks()">**

**Double click on me**

**</a>**

**<script>**

**function geeks(){**

**document.getElementById("gfg")**

**.innerHTML='Welcome to GeeksforGeeks';**

**}**

**</script>**

**<p id="gfg"></p>**

**Q-16. • How can a page be forced to load another page in JavaScript?**

**In this post, we'll look at how JavaScript can be used to compel one page to load another.**

**Approach: To forcefully load another page in Javascript, we can use the window.location attribute inside the script element. It is a pointer to a Location object, which stands in for the document's current location. By visiting a window, we can modify its URL.**

**Syntax:**

**<script>**

**window.location = <Path / URL>**

**</script>**

**Example:**

**<script>**

**window.location = "https://www.geeksforgeeks.org/"**

**</script>**

**In the example above, we can see that modifying the window.location object inside of Javascript allows us to forcefully load any page from Javascript without using a href tag. We'll create a simple working example so we can learn it on the job.**

**Here is how it is done step by step:**

**Step 1. Make a file called index.html as the first step. Add two buttons and a heading to it. One button forces the loading of a live URL while the other forces the loading of a local HTML page. Two functions are contained in the script> tag; one loads the gfg homepage, and the other uses the window.location property to load a local HTML page.**

**<!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\_wikipedia()">**

**Force Load wikipedia Page**

**</button>**

**<br><br>**

**<button onclick="force\_load\_local()">**

**Force Load Local HTML page**

**</button>**

**<script>**

**function force\_load\_wekipedia() {**

**window.location =**

**"**[**https://www.wikipedia.org/**](https://www.wikipedia.org/)**"**

**}**

**function force\_load\_local() {**

**window.location =**

**"F:/gfg/PageRedirect/newPage.html"**

**}**

**</script>**

**</body>**

**</html**>

**Step 2: Make a newPage.html file. This is the local HTML page that Javascript would load.**

**<!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">**

**<title> New Page </title>**

**</head>**

**<body>**

**<h3>This is the new loaded page</h3>**

**</body>**

**</html>**

**Q – 17. • What are the disadvantages of using innerHTML in JavaScript?**

**Step The Document Object Model (DOM)'s innerHTML attribute is used to set or return an element's HTML content. where the textual content of the HTML element is represented by the return value. It enables JavaScript code to alter a website that is currently being shown. More specifically, it alters or restores an element's internal HTML information. Since it is the simplest approach to manipulate DOM, the innerHTML property is frequently used to alter the contents of webpages. But utilizing innerHTML in JavaScript has several drawbacks.**

**Using JavaScript's innerHTML property has the following drawbacks:**

**The process of employing innerHTML is more slower since its contents are constructed gradually and because previously parsed elements and contents must be re-parsed, which takes time.**

**keeps any DOM element's event handlers in tact: The new elements produced by automatically setting innerHTML do not acquire event handlers. To do this, one must manually attach the event handlers to new elements and maintain track of the event handlers. On some browsers, this can result in a memory leak.**

**All content has been changed: When you use innerHTML to add, append, delete, or edit material on a webpage, all of the content and any DOM nodes inside that element are replaced and recreated.**

**InnerHTML appending is not supported: In JavaScript, adding is often done with the += operator. However, when using innerHTML to append to an HTML tag, the entire tag is re-parsed.**

**Example:**

**<p id="geek">Geeks</p>**

**title = document.getElementById('#geek')**

**// The whole "geek" tag is reparsed**

**title.innerHTML += '<p> forGeeks </p>'**

**Problem: The old content gets overwritten even while the item is present.object innerHTML is used.Instead of using object.innerHTML += 'html', use innerHTML + 'html'. Appending is impossible without repartitioning the entire innerHTML. As a result, using innerHTML becomes extremely sluggish. When dynamic DOM elements must be formed, string concatenation simply cannot scale since it becomes challenging to keep track of the plus signs and quotation marks' openings and closings.**

**Can cause the document to be broken: Since innerHTML does not offer appropriate validation, any legitimate HTML code can be used. The JavaScript document might be broken as a result. Even flawed HTML can be utilized, which could result in unpleasant surprises.**

**The ability of innerHTML to add text and components to the webpage makes it possible for malevolent users to change and display unwanted or destructive content within other HTML element tags. This is known as cross-site scripting (XSS). Additionally, critical information may be lost, leaked, or changed as a result of cross-site scripting.**

<!DOCTYPE html>

<html>

<head>

    <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>

**Q – 18. Create password field with show hide functionalities**

**This post will demonstrate how to use JavaScript to hide and reveal passwords. When filling out a form, there may be instances where we need to type a password and want to review our previous input. There is a checkbox that, when selected, makes the characters visible so that you can see that. In this lesson, we'll use JavaScript to implement the toggle password feature.**

**Approach:**

1. **Create an HTML form with a password-only input field.**
2. **Make a checkbox that will serve as the toggle.**
3. **Make a function that responds to the checkbox being clicked by the user.**

**Example:**

Password is Welcome12@.

So, on typing it will show like this **\*\*\*\*\*\*\*\*\***

And on clicking the checkbox it will show the characters: Welcome12@.

|  |
| --- |
| <!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">      <title>          Show and hide password using JavaScript      </title>  </head>    <body>      <strong>          <p>Click on the checkbox to show              or hide password:          </p>      </strong>        <strong>Password</strong>:      <input type="password" value="geeksforgeeks"                id="typepass">      <input type="checkbox" onclick="Toggle()">      <strong>Show Password</strong>        <script>          // Change the type of input to password or text          function Toggle() {              let temp = document.getElementById("typepass");                if (temp.type === "password") {                  temp.type = "text";              }              else {                  temp.type = "password";              }          }      </script>  </body>  </html> |