

Q: 1

**What is JavaScript?**

JavaScript is a programming language that allows you to **create interactive** and **dynamic** web pages:

**Functionality**: JavaScript enables you to add features like animated graphics, interactive maps, and content updates to web pages.

**User interaction**: JavaScript allows browsers to respond to user interactions and change the layout of a web page.

**Development**: JavaScript can be used for both client-side and server-side development.

**Tools**: Developers have created tools on top of JavaScript, such as APIs, frameworks, and libraries, to make it easier to build sites and applications.

**Game development**: JavaScript can be used to build games with HTML5 and an API like WebGL.

JavaScript is a powerful programming language that can add interactivity to a website. It was invented by Brendan Eich. JavaScript is versatile and beginner-friendly. With more experience, you'll be able to create games, animated 2D and 3D graphics, comprehensive database-driven apps, and much more!

Q: 2

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

isNaN() returns true if a number is Not-a-Number. In other words: isNaN() converts the value to a number before testing it.

Ex:

<html>

<body>

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

<script>

let some = "Hello";

document.getElementById("ex").innerHTML = Number.isNaN(some);

</script>

</body>

</html>

Output : **false**

Q: 3

**What is negative Infinity?**

NEGATIVE\_INFINITY is a special numeric value that is returned when an arithmetic operation or mathematical function generates a negative value greater than the largest representable number in JavaScript (i.e., more negative than -Number. MAX\_VALUE). JavaScript displays the NEGATIVE\_INFINITY value as -Infinity.

Ex:

<html>

<body>

<p id="ex"></p>

<script>

let n = (-24) \* 2;

document.getElementById("ex").innerHTML = n;

</script>

</body>

</html>

Output: **-48**

Q: 4

**Which company developed JavaScript?**

JavaScript was created at Netscape Communications by Brendan Eich in 1995. Netscape and Eich designed JavaScript as a scripting language for use with the company's flagship web browser, Netscape Navigator.

It was created by Brendan Eich while he was working at Netscape. JavaScript was initially designed to add interactivity to web pages and is now a core technology of the World Wide Web, alongside HTML and CSS.

It was developed for **Netscape 2**, and became the **ECMA-262** standard in 1997.

In 1996, Netscape and Brendan Eich took JavaScript to the ECMA international standards organization, and a technical committee (TC39) was created to develop the language.

ECMA-262 Edition 1 was released in June 1997.

|  |  |  |
| --- | --- | --- |
| 1995 |  | JavaScript was invented by Brendan Eich |
| 1996 |  | Netscape 2 was released with JavaScript 1.0 |
| 1997 |  | JavaScript became an ECMA standard (ECMA-262) |
| 1997 | ES1 | ECMAScript 1 was released |
| 1997 | ES1 | IE 4 was the first browser to support ES1 |
| 1998 | ES2 | ECMAScript 2 was released |
| 1998 |  | Netscape 42 was released with JavaScript 1.3 |
| 1999 | ES2 | IE 5 was the first browser to support ES2 |
| 1999 | ES3 | ECMAScript 3 was released |
| 2000 | ES3 | IE 5.5 was the first browser to support ES3 |
| 2000 |  | Netscape 62 was released with JavaScript 1.5 |
| 2000 |  | Firefox 1 was released with JavaScript 1.5 |
| 2008 | ES4 | ECMAScript 4 was abandoned |
| 2009 | ES5 | ECMAScript 5 was released |
| 2011 | ES5 | IE 9 was the first browser to support ES5 \* |
| 2011 | ES5 | Firefox 4 was released with JavaScript 1.8.5 |
| 2012 | ES5 | Full support for ES5 in Safari 6 |
| 2012 | ES5 | Full support for ES5 in IE 10 |
| 2012 | ES5 | Full support for ES5 in Chrome 23 |
| 2013 | ES5 | Full support for ES5 in Firefox 21 |
| 2013 | ES5 | Full support for ES5 in Opera 15 |
| 2014 | ES5 | Full support for ES5 in all browsers |
| 2015 | ES6 | ECMAScript 6 was released |
| 2016 | ES6 | Full support for ES6 in Chrome 51 |
| 2016 | ES6 | Full support for ES6 in Opera 38 |
| 2016 | ES6 | Full support for ES6 in Safari 10 |
| 2017 | ES6 | Full support for ES6 in Firefox 54 |
| 2017 | ES6 | Full support for ES6 in Edge 15 |
| 2018 | ES6 | Full support for ES6 in all browsers \*\* |

Q: 5

**What are undeclared and undefined variables?**

In JavaScript, the main difference between an undeclared variable and an undefined variable is how the variable was declared or assigned a value:

**Undeclared**

A variable is undeclared if it was not declared using the appropriate keyword, such as var, let, or const. Accessing an undeclared variable will result in a Reference Error. For example, test Var = "This is undeclared" is an undeclared variable.

**Undefined**

A variable is undefined if it has not been assigned a value. undefined is a primitive data type in JavaScript that represents the absence of a value. For example, let x; console.log(x); // undefined is an undefined variable.

Here are some other differences between undeclared, undefined, and null variables:

**Null**

A variable is null when it has been assigned a value of null, which represents the intentional absence of a value. The type of a null value is object.

**Falsy**

Both null and undefined are considered falsy values, meaning they represent an absence of some value type. In an if-else statement, an undefined variable will follow the same flow as any other condition where the variable has a value of false.

Ex:

let num1;

console.log(num1);

console.log(num2);

Output:

ReferenceError: num1 is undefined.

ReferenceError: num2 is not defined.

Q: 6

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

Creation of new element: New elements can be created in JS by using the **createElement()** method.

Syntax:

document.createElement("<*tagName*>");

// Where <*tagName*> can be any HTML

// tagName like div, ul, button, etc.

// newDiv element has been created

For Eg: **let newDiv = document.createElement("div");**

Ex:

<script>

let **newEle** = document.createElement('**div**');

newEle.style.cssText =`height:300px; width:300px; margin:50px

auto; background-color:red;`;

document.body.append(newEle);

</script>

Output:

Q: 7

**What is the difference between View State and Session State?**

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 post back operation occurring.

* **Session State:** It can be used to store information that you wish to access on different web pages.
* **View State** It can be used to store information that you wish to access from same web page.

| **View State** | **Session State** |
| --- | --- |
| 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 post back 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. |
| View State values are lost/cleared when new page is loaded. | Session State can be cleared by programmer or user or in case of timeouts. |

Q: 8

**What is === operator?**

In JavaScript, **=== is a strict equality operator**. It checks if two values are equal and also ensures they are of the same type.

The triple equals (===) are a comparison operator used in programming languages like JavaScript. It checks whether the operands are equal in both value and type. Unlike the double equals (==) operator, which performs type coercion before comparison, the triple equals operator does not perform any type casting.

Ex:

let a = 5;

let b = '5';

console. log (**a === b**);

**false** because **one is a number** and the **other is a string**.

Q: 9

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

Use the document. query Selector () method to select that element by passing its CSS selector as an argument. This method returns the first element in the document that matches the selector. Access the class list property of the selected element to manipulate its classes.

Ex:

**By Style**

<script>

let newEle = document.createElement('div');

**newEle.style.cssText =`height:300px; width:300px; margin:50px auto; background-color:red;`**

document.body.append(newEle);

</script>

**By Class**

<html>

<head><style>

.box1

{

        height: 300px;

        width: 300px;

        background-color: green;

        margin: 50px auto;

}

</style></head>

<body>

<script>

      let newEle = document.createElement("div");

      newEle.classList.add("box1");

      document.body.append(newEle);

    </script>

  </body>

</html>

Output :

Q: 10

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

It can be used to perform operations such as reading, writing, updating, and deleting files.

The fs. **readFile() and rs. writeFile()** methods are used to read and write of a file using java script.

**fs.readFile()**

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 )

**fs.writeFile()**

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 )

Q: 11

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

**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.

Syntax:

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

Ex:

<html>

<body>

<p id="ex"></p>

<script>

let newStr = "";

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

**{**

**newStr += "The number is " + i + "<br>";**

**}**

document.getElementById("ex").innerHTML = newStr;

</script>

</body>

</html>

Output:

The number is 0  
The number is 1  
The number is 2  
The number is 3  
The number is 4

The number is 5  
The number is 6  
The number is 7  
The number is 8  
The number is 9

Q: 12

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

There's a function called **parseInt()** in JavaScript, this is used for parsing a string as an argument and it returns an integer of the specified radix (basically the base of the numerical system) as output.

Ex:

<p id="ex"></p>

<script>

document.getElementById("ex").innerHTML =

parseInt("10") + "<br>" +

parseInt("10.00") + "<br>" +

parseInt("10.33") + "<br>" +

parseInt("34 45 66") + "<br>" +

parseInt(" 60 ");

</script>

Output:

**10  
10  
10  
34  
60**

Q: 13

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

The delete operator removes a property from an object. If the property's value is an object and there are no more references to the object, the object held by that property is eventually released automatically**.**

**Syntax**

delete object

delete object. property

delete object['property']

Ex:

**let** person =

{

name: "Jay",

surname: "Dholakia",

age: 24

}

console.log (**delete** person. age);

console.log(person);

Q: 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**.

**Alert Box:**

<button onclick="myFunction()">Alert Box</button>

<script>

function myFunction() {

**alert ("Hello Good Morning! …");**

}

</script>

**Confirm Box:**

<button onclick="myFunction()">Try it</button>

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

<script>

function myFunction()

{

var txt;

if (confirm("Press a button!")) {

txt = "You pressed OK!";

} else {

txt = "You pressed Cancel!";

}

document.getElementById("demo").innerHTML = txt;

}

</script>

**Prompt box**.

prompt("Please enter your name", "Harry Potter");

Q: 15

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

JavaScript void 0 means returning undefined (void) as a primitive value.

You might come across the term “JavaScript: void (0)” while going through HTML documents.

It is used to prevent any side effects caused while inserting an expression in a web page.

The void operator evaluates an expression and returns undefined. By running void (0) in the URL JavaScript code, nothing is evaluated or returned.

Of all the unary operators, void offers the best semantic, because it clearly signals that the return value of the function invocation should be discarded.

Ex:

function emptyBox() {

return void(0);

}

console.log(emptyBox());

Output:

Undefined

Q: 16

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

In JavaScript, you can force a page to load another page using the window.location object.

This can be done in several ways, including setting window.location.**href,** using window.location.**assign()**, or using window.location.**replace()**.

Ex:

**href:**

function goToNextPage()

{

**window.location.href** = 'https://www.nextpage.html';

}

goToNextPage();

**assign:**

function goToNexPage()

{

**window.location.assign**

('https://www.example.com');

}

goToNexPage();

**replace:**

function goToNextPage()

{ **window.location.replace**('https://www.example.com');

}

goToNextPage();

Q: 17

**What are the disadvantages of using inner HTML in JavaScript?**

It is very slow because as inner HTML already parses the content even, we have to parse the content again so that's why it takes time. When we have used the event handlers then the event handlers are not automatically attached to the new elements created by inner HTML.

Event handlers attached to any DOM element are preserved

When we have used the event handlers then the event handlers are not automatically attached to the new elements created by inner HTML. To change that, we have to track the event handlers and manually attach them to a new element.

It means that first, we have to fetch the element property through inner HTML, and then we have to attach them to a new element.

**It is not possible to append inner HTML,**

In JavaScript, ‘+=’ is commonly used for appending. However, when using inner HTML to append to an HTML tag, the entire tag is re-parsed.

**Breaks the document,**

Inner HTML does not provide proper validation, so any valid HTML code can be used. This has the potential to break the JavaScript document. Even broken HTML can be used, which can cause unexpected issues.