1. JavaScript Can Change HTML Content

```
<p id="demo">JavaScript can change HTML content.</p>

<button type="button"
onclick="document.getElementById('demo').innerHTML='hello
world'">click</button>
```

2.to change image path by clicking a button

```
onclick="document.getElementById('myImage').src='pic_bulbon.gif'"
```

3.JavaScript Can Change HTML Styles (CSS)

```
onclick="document.getElementById('demo').style.fontSize='100px'"
```

4.JavaScript Can Hide HTML Elements

```
onclick="document.getElementById('demo').style.display='none'"
```

5.javaScript functions

```
function Function(){
        document.getElementById("demo").innerHTML="you called it";
}
<button onclick="Function()">call it</button>
```

6.external file

```
<script src="myScript.js"></script>
```

7.To add several script files to one page

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

8.JavaScript Display Possibilities

Writing into an HTML element, using `innerHTML`.

- Writing into the HTML output using `document.write()`.
- Writing into an alert box, using `window.alert()`.
- Writing into the browser console, using `console.log()`.

9.using document.write()

```
<button onclick="document.write(90+100)">add</button>
```

10.using window.alert()

```
window.alert("see");
or
alert(“see); we can skip window
```

11.using console.log()

```
console.log(90);
```

12.JavaScript Print

```
<button onclick="window.print()">Print this page</button>
```

13.keywords

Keyword	Description
var	Declares a variable
let	Declares a block variable
const	Declares a block constant
if	Marks a block of statements to be executed on a condition
switch	Marks a block of statements to be executed in different cases
for	Marks a block of statements to be executed in a loop
function	Declares a function
return	Exits a function
try	Implements error handling to a block of statements

14.JavaScript Variables can be declared in 4 ways:

Automatically

Using `var`

Using `let`

Using `const`

15.Automatically

```
x = 5;
y = 6;
z = x + y;
```

In this example, x, y, and z are undeclared.
They are automatically declared when first used.

16.variables

After the declaration, the variable has no value (technically it is `undefined`).

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

<script>
let carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
```

17.Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable declared with `var`, it will not lose its value.
The variable `carName` will still have the value "Volvo" after the execution of these statements:

```
var carName = "Volvo";
var carName;
```

18.let and const

You cannot re-declare a variable declared with `let` or `const`.

This will not work:

```
let carName = "Volvo";
let carName;
```

19.Since JavaScript treats a dollar sign as a letter, identifiers containing $ are valid variable names,Since JavaScript treats underscore as a letter, identifiers containing _ are valid variable names:

```
let $ = "Hello World";
let $$$ = 2;
let $myMoney = 5;
let _x = 2;
```

20.let

The `let` keyword was introduced in ES6 (2015)

Variables declared with `let` have **Block Scope**

Variables declared with `let` must be **Declared** before use

Variables declared with `let` cannot be **Redeclared** in the same scope

21.

Before ES6 (2015), JavaScript did not have **Block Scope**.
JavaScript had **Global Scope** and **Function Scope**.

ES6 introduced the two new JavaScript keywords: `let` and `const`.

These two keywords provideed **Block Scope** in JavaScript:

```
{
let x = 2;
}
// x can NOT be used here
```

22.Variables declared with the `var` always have **Global Scope**.

```
{
var x = 2;
}
// x CAN be used here
```

23.Cannot be Redeclared

With `let` you **can not** do this:

```
let x = "John Doe";

let x = 0;
```

Variables defined with `var` **can** be redeclared.

With `var` you **can** do this:

```
var x = "John Doe";
var x = 0;
```

24.Redeclaring Variables

Redeclaring a variable using the `var` keyword can impose problems.

Redeclaring a variable inside a block will also redeclare the variable outside the block:

```
var x = 10;
// Here x is 10

{
var x = 2;
// Here x is 2
}

// Here x is 2
```

25.Redeclaring a variable using the `let` keyword can solve this problem.

```
let x = 10;
// Here x is 10

{
let x = 2;
// Here x is 2
}

// Here x is 10
```

26.With `let`, redeclaring a variable in the same block is NOT allowed:

```
var x = 2;   // Allowed
let x = 3;   // Not allowed

{
let x = 2;   // Allowed
let x = 3;   // Not allowed
}

{
let x = 2;   // Allowed
var x = 3;   // Not allowed
}
```

27.Redeclaring a variable with `let`, in another block, IS allowed:

```
let x = 2;   // Allowed

{
let x = 3;   // Allowed
}

{
let x = 4;    // Allowed
}
```

28.Let Hoisting

Variables defined with `var` are **hoisted** to the top and can be initialized at any time.
Meaning: You can use the variable before it is declared:

```
carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
var carName;
o/p:Volvo
```

29.Variables defined with `let` are also hoisted to the top of the block, but not initialized.
Meaning: Using a `let` variable before it is declared will result in a `ReferenceError`:

```
try{
carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
let carName;}
catch(err)
{
document.write(err);
}
o/p:ReferenceError: can't access lexical declaration 'carName' before initialization
```

30.const

The `const` keyword was introduced in ES6 (2015)

Variables defined with `const` cannot be **Redeclared**

Variables defined with `const` cannot be **Reassigned**

Variables defined with `const` have **Block Scope**

JavaScript `const` variables must be assigned a value when they are declared:

```
const PI = 3.14159265359; //allowed
const p;//not allowed
```

The keyword `const` is a little misleading.

It does not define a constant value. It defines a constant reference to a value.

Because of this you can NOT:

- Reassign a constant value
- Reassign a constant array
- Reassign a constant object

But you CAN:

- Change the elements of constant array
- Change the properties of constant object

```
// You can create a constant array:
const cars = ["Saab", "Volvo", "BMW"];

// You can change an element:
cars[0] = "Toyota";

// You can add an element:
cars.push("Audi");
```

But you can NOT reassign the array:

```
const cars = ["Saab", "Volvo", "BMW"];

cars = ["Toyota", "Volvo", "Audi"];    // ERROR
```

31.You can change the properties of a constant object:

```
// You can create a const object:
const car = {type:"Fiat", model:"500", color:"white"};

// You can change a property:
car.color = "red";

// You can add a property:
car.owner = "Johnson";
```

But you can NOT reassign the object:

Example

```
const car = {type:"Fiat", model:"500", color:"white"};

car = {type:"Volvo", model:"EX60", color:"red"};    // ERROR
```

```
const x = 10;
// Here x is 10

{
const x = 2;
// Here x is 2
}
// Here x is 10
```

Redeclaring an existing `var` or `let` variable to `const`, in the same scope, is not allowed:

```
var x = 2;     // Allowed
const x = 2;   // Not allowed
{
let x = 2;     // Allowed
const x = 2;   // Not allowed
}
{
const x = 2;   // Allowed
const x = 2;   // Not allowed
}
```

32.Reassigning an existing `const` variable, in the same scope, is not allowed:

```
        const x = 2;     // Allowed
x = 2;          // Not allowed
var x = 2;      // Not allowed
let x = 2;      // Not allowed
const x = 2;    // Not allowed
{
  const x = 2;   // Allowed
  x = 2;         // Not allowed
  var x = 2;     // Not allowed
  let x = 2;     // Not allowed
  const x = 2;   // Not allowed
}
```

Redeclaring a variable with `const`, in another scope, or in another block, is allowed:

```
const x = 2;       // Allowed
{
  const x = 3;   // Allowed
}
{
  const x = 4;   // Allowed
}
```

33.hoisting

Variables defined with `const` are also hoisted to the top, but not initialized.

Meaning: Using a `const` variable before it is declared will result in a `ReferenceError`:

```
        try {
  alert(carName);
  const carName = "Volvo";
}
catch (err) {
  document.getElementById("demo").innerHTML = err;
}
```

o/p:With **const**, you cannot use a variable before it is declared:

ReferenceError: can't access lexical declaration 'carName' before initialization

34.operators

** Exponentiation (ES2016)

Note that strings are compared alphabetically:

```
let text1 = "58";
let text2 = "581";
let result = text1 < text2; //true
let text1 = "581";
let text2 = "58";
let result = text1 < text2;//false
let text1 = "881";
let text2 = "98";
let result = text1 < text2;//true
```

```
let text1 = "20";
let text2 = "5";
let result = text1 < text2; //true comaparison between ascii of 2 &5
let text1 = "50";
let text2 = "5";
let result = text1 < text2;//false 5 and 5 then go to next letter
let text1 = "ABC";
let text2 = "AB";
let result = text1 < text2;//false
let text1 = "AB";
let text2 = "ABC";
let result = text1 < text2;//true
```

35.The ??= Operator

The **Nullish coalescing assignment operator** is used between two values.

If the first value is undefined or null, the second value is assigned.

The `??=` operator is an ES2020 feature.

```
let x;
document.getElementById("demo").innerHTML = x ??= 5;
```

o/p:The ??= operator is used between two values. If the first value is undefined or null, the second value is assigned.

5

36.Datatypes

JavaScript has 8 Datatypes

1. String
2. Number
3. Bigint

4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

The Object Datatype

The object data type can contain:

1. An object
2. An array
3. A date

```
// Booleans
let x = true;
let y = false;

// Object:
const person = {firstName:"John", lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo", "BMW"];

// Date object:
const date = new Date("2022-03-25");
```

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

```
let x;// Now x is undefined
x = 5;// Now x is a Number
x = "John"; // Now x is a String
```

Extra large or extra small numbers can be written with scientific (exponential) notation:

```
let y = 123e5;    // 12300000
let z = 123e-5;   // 0.00123
```

37.JavaScript BigInt

Javascript numbers are always one type:
double (64-bit floating point).
You cannot perform math between a BigInt type and a Number type.

All JavaScript numbers are stored in a a 64-bit floating-point format.

JavaScript BigInt is a new datatype (ES2020) that can be used to store integer values that are too big to be represented by a normal JavaScript Number.

Example

```
let x = BigInt("12345678901234567890123456789O");
```

38.Functions

Accessing a function without () returns the function and not the function result:

```
function toCelsius(f) {
  return (5/9) * (f-32);
}

let value = toCelsius;
document.getElementById("demo").innerHTML = value;
o/p:function toCelsius(f) { return (5/9) * (f-32); }
```

Accessing a function with incorrect parameters can return an incorrect answer:

```
function toCelsius(f) {
  return (5/9) * (f-32);
}

let value = toCelsius();
document.getElementById("demo").innerHTML = value;
o/p:NaN
```

As you see from the examples above, `toCelsius` refers to the function object, and `toCelsius()` refers to the function result.

38.object

Objects can also have **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

```
const person = {
firstName: "John",
lastName : "Doe",
id:5566,
 fullName : function() {
 return this.firstName + " " + this.lastName;
}
};
```

38.THIS

the `this` keyword refers to an **object**.

In an object method, `this` refers to the **object**.
Alone, `this` refers to the **global object**.
In a function, `this` refers to the **global object**.
In a function, in strict mode, `this` is `undefined`.
In an event, `this` refers to the **element** that received the event.
Methods like `call()`, `apply()`, and `bind()` can refer `this` to **any object**.

39.Events

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

40.Strings

When using the == operator, x and y are **equal**:

```
let x = "John";
let y = new String("John");
```

When using the === operator, x and y are **not equal**:

```
let x = "John";
let y = new String("John");
```

JavaScript objects cannot be compared.
Comparing two JavaScript objects **always** returns **false**.

```
let x = new String("John");  // x is an object
let y = new String("John");  // y is an object
document.getElementById("demo").innerHTML = (x===y);//false
let x = new String("John");  // x is an object
let y = new String("John");  // y is an object
document.getElementById("demo").innerHTML = (x==y);//false
```