ES6 Interview Questions

A list of frequently asked **ES6 Interview Questions** and Answers are given below.

1) What is ES6 or ECMAScript 2015?

ES6 was released in June 2015, which is stated as the sixth edition of the language. Initially, it was named **ES6** and later renamed to ECMAScript 2015. This edition includes several new features that are modules, iterators, class, arrow functions, for...of loop, promises, and many more. Brendan Eich developed it.

---

2) Define ECMAScript.

It is the specification that is defined in the ECMA-262 standard to create a general-purpose scripting language.

---

3) What are the new features introduced in ES6?

The new features that are introduced in ES6 are listed as follows:

- Let and const keywords.
- Default Parameters.
- Arrow functions.
- Template Literals.
- Object Literals.
- Rest and spread operators.
- Destructuring assignment.
- Modules, Classes, Generators, and iterators.

- Promises, and many more.

---

4) Define let and const keywords.

let: The variables declared using **let** keyword will be mutable, i.e., the values of the variable can be changed. It is similar to **var** keyword except that it provides block scoping.

const: The variables declared using **the const** keyword are immutable and block-scoped. The value of the variables cannot be changed or re-assigned if they are declared by using **the const** keyword.

---

5) What is the arrow function, and how to create it?

Arrow functions are introduced in ES6. Arrow functions are the shorthand notation to write ES6 functions. The definition of the arrow function consists of parameters, followed by an arrow (=>) and the body of the function.

An Arrow function is also called as **'fat arrow'** function. We cannot use them as constructors.

Syntax

```
1. const functionName = (arg1, arg2, ...) => {
2.     //body of the function
3. }
```

---

6) Give an example of an Arrow function in ES6? List down its advantages.

Arrow function provides us a more accurate way of writing the functions in JavaScript. They allow us to write smaller function syntax.

The context within the arrow functions is lexically or statically scoped. Arrow functions do not include any prototype property, and cannot be used with the new keyword.

You can learn more about arrow functions by clicking on this link ES6 Arrow Function.

Example

```
1. var myfun = () => {
```

```
2.      console.log("It is an Arrow Function");
3.     };
4.     myfun();
```

Output

```
It is an Arrow Function
```

Advantages of Arrow Function

The advantages of the arrow function are listed below:

- It reduces code size.
- The return statement is optional for a single line function.
- Lexically bind the context.
- Functional braces are optional for a single-line statement.

---

7) Discuss spread operator in ES6 with an example.

The spread operator is represented by three dots (...) to obtain the list of parameters. It allows the expansion of an iterable such as array or string in places where more than zero arguments are expected.

The spread operator syntax is similar to the rest operator, but functionality is entirely opposite to it. It is also used to combine or to perform the concatenation between arrays. Let's understand it by an example.

Example

```
1. let num1 = [40,50,60];
2.
3. let num2 = [10,20,30,...num1,70,80,90,100];
4.
5. console.log(num2);
```

Output

```
[
  10, 20, 30, 40,  50,
  60, 70, 80, 90, 100
]
```

---

8) Discuss the Rest parameter in ES6 with an example.

It is introduced in ES6 that improves the ability to handle the parameters. With rest parameters, it is possible to represent indefinite parameters as an array. By using the rest parameter, we can call a function with any number of arguments.

Example

```
function show(...args) {
    let sum = 0;
    for (let i of args) {
        sum += i;
    }
    console.log("Sum = "+sum);
}

show(10, 20, 30);
```

Output

```
Sum = 60
```

---

9) What are the template literals in ES6?

Template literals are a new feature introduced in ES6. It provides an easy way of creating multiline strings and perform string interpolation.

Template literals allow embedded expressions and also called as string literals.

Prior to ES6, template literals were referred to as **template strings**. Template literals are enclosed by the **backtick (` `) character**. Placeholders in template literals are represented by the dollar sign and the curly braces **(${expression})**. If we require to use an expression within the backticks, then we can place that expression in the **(${expression})**.

To learn more about template literals in ES6, follow this link ES6 Template Literals.

Example

```
let str1 = "Hello";

let str2 = "World";

let str = `${str1} ${str2}`;
console.log(str);
```

Output

```
Hello World
```

---

10) Discuss Destructuring Assignment in ES6.

Destructuring is introduced in ECMAScript 2015 or ES6 to extract data from objects and arrays into separate variables. It allows us to extract smaller fragments from objects and arrays.

To learn more about array destructuring in ES6, follow this link ES6 Array Destructuring.

To learn more about object destructuring in ES6, follow this link ES6 Object Destructuring.

Example

```
let fullname =['Alan','Rickman'];
let [fname,lname] = fullname;
console.log (fname,lname);
```

Output

```
Alan Rickman
```

---

11) How to create a class in ES6?

This keyword is used for creating the class. We can include the classes in our code either by using class expression or by class declaration. A class definition can only include **functions** and **constructors**. These components are together called as data members of the class.

Constructors in classes allocate the memory to the objects of the class. Functions in a class are responsible for performing the actions to the objects.

To learn more about classes in ES6, follow this link ES6 Classes.

Let us see the syntax for creating classes.

Syntax: In ES5

```
var var_name = new class_name {
}
```

Syntax: In ES6 (Using class keyword)

```
class class_name{
```

```
2. }
```

---

12) What do you understand by Generator function?

A generator provides us a new way to work with iterators and functions. The generator is a special kind of function that may be paused in the middle either one or many times and can be resumed later. The declaration **function* (*function* keyword followed by an asterisk)** is used to define a generator function.

When the generator gets called, it does not run its code. Instead, it returns a special object, which is called a **Generator object** to manage the execution. Let us see an example of generators in ES6.

To learn more about Generators in ES6, follow this link ES6 Generators.

Example

```
1. function* gen()
2. {
3. yield 100;
4. yield;
5. yield 200;
6. }
7. // Calling the Generator Function
8. var mygen = gen();
9. console.log(mygen.next().value);
10.console.log(mygen.next().value);
11.console.log(mygen.next().value);
```

Output

```
100
undefined
200
```

---

13) What are the default parameters?

By using the default parameters, we can initialize named parameters with default values if there is no value or **undefined** is passed.

Example

```
1. var show = (a, b=200) => {
```

```
    console.log(a + " " + b);
}
show(100);
```

Output

```
100 200
```

---

14) What do you mean by IIFE (Immediately invoked function expressions)?

IIFE is a function in JavaScript that runs as soon as it is defined. It is also called as the **Self-Executing Anonymous Function**. It includes two major parts that are as follows:

- The first part is an anonymous function that has a lexical scope (static scope), which is enclosed within the **Grouping operator ()**.
- The second part creates the IIFE by which the JavaScript engine will interpret the function directly.

You can learn more about arrow functions by clicking on this link ES6 IIFE.

Example

```
(function()
 {
 console.log("Hello World");
})();
```

Output

```
Hello World
```

---

15) Discuss the for...in loop.

It is similar to for loop that iterates through the properties of an object. It is useful when we require to visit the properties or keys of the object.

Example

```
function Mobile(model_no){
    this.Model = model_no;
    this.Color = 'White';
```

```
    this.RAM = '4GB';
    }
    var Samsung = new Mobile("Galaxy");
    for(var props in Samsung)
    {
    console.log(props+ " : " +Samsung[props]);
    }
```

Output

```
Model: Galaxy
Color:  White
RAM: 4GB
```

---

16) Discuss the for...of loop.

This loop is used for iterating the iterables (arrays, string, etc.).

Example

```
var fruits = ['Apple', 'Banana', 'Mango', 'Orange'];
for(let value of fruits)
{
  console.log(value);
}
```

Output

```
Apple
Banana
Mango
Orange
```

---

17) Define set.

A set is a data structure that allows us to create a collection of unique values. It is a collection of values that are similar to arrays, but it does not include any duplicates. It supports both object references and primitive values.

To learn more about Sets in ES6, follow this link ES6 Sets.

Example

```
let colors = new Set(['Green', 'Red', 'Orange', 'Yellow', 'Red']);
```

2. console.log(colors);

Output

```
Set { 'Green', 'Red', 'Orange', 'Yellow' }
```

---

18) Define Map.

Prior to ES6, when we require the mapping of keys and values, we often use an object. **Map object** is a new collection type, which is introduced in ES6. It holds the key-value pairs in which any type of values can be used as either keys or values.

A map object always remembers the actual insertion order of the keys. Maps are ordered, so they traverse the elements in their insertion order.

To learn more about Map in ES6, follow this link ES6 Maps.

---

19) What do you understand by Weakset?

Using weakset, it is possible to store weakly held objects in a collection. As similar to set, weakset cannot store duplicate values. Weakset cannot be iterated.

Weakset only includes **add(value), delete(value)** and **has(value)** methods of the set object.

---

20) What do you understand by Weakmap?

Weak maps are almost similar to maps, but the keys in weak maps must be objects. It stores each element as a key-value pair where keys are weakly referenced. Here, the keys are objects, and the values are arbitrary.

A weak map object iterates the element in their insertion order. It only includes **delete(key), get(key), has(key)** and **set(key, value)** method.

---

21) Explain Promises in ES6.

ES6 promises are the easiest way to work with asynchronous programming in JavaScript. Asynchronous programming includes running of processes individually from the main thread, and it notifies the main thread when it gets complete.

Prior to ES6, there is the use of **Callbacks** for performing asynchronous programming. Promises are used to overcome the problem of **Callback hell**.

To learn more about promises, follow this link: ES6 Promises.

---

22) What are the states of promises in ES6?

Promises have mainly three states that are as follows:

- **Pending:** It is the initial state of every promise. It represents that the result has not been computed yet.
- **Fulfilled:** It represents the completion of an operation.
- **Rejected:** It represents the failure that occurs during computation.

Once the promise is fulfilled or rejected, then it will be immutable.
The **Promise()** constructor takes two arguments that are **rejected** function and a **resolve** function. Based on the asynchronous operation, it returns either the first argument or the second argument.

---

23) What do you understand by Callback and Callback hell in JavaScript?

Callback: It is used to handle the execution of function after the completion of the execution of another function. A callback would be helpful in working with events. In the callback, a function can be passed as an argument to another function. It is a great way when we are dealing with basic cases such as minimal asynchronous operations.

Callback hell: When we develop a web application that includes a lot of code, then working with callback is messy. This excessive Callback nesting is often referred to as **Callback hell**.

---

24) List the comparisons between ES5 and ES6.

ES5 and ES6 are similar in their nature, but there are some differences between them. The comparison between ES5 and ES6 are tabulated as follows:

Based on	ES5	ES6
Definition	ES5 is the fifth edition of the ECMAScript (a trademarked scripting language specification defined by ECMA International)	ES6 is the sixth e (a trademarked s

		specification defi International).
Release	It was introduced in 2009.	It was introduced
Data-types	ES5 supports primitive data types that are **string, boolean, number, null,** and **undefined**.	In ES6, there are JavaScript data t primitive data ty supporting uniqu
Defining Variables	In ES5, we could only define the variables by using the **var** keyword.	In ES6, there are variables that are
Performance	As ES5 is prior to ES6, there is a non-presence of some features, so it has a lower performance than ES6.	Because of new f shorthand storag a higher perform
Support	A wide range of communities supports it.	It also has a lot o it is lesser than E
Object Manipulation	ES5 is time-consuming than ES6.	Due to destructu object manipulat more smoothly in
Arrow Functions	In ES5, both **function** and **return** keywords are used to define a function.	An arrow functio introduced in ES require the **funct** the function.
Loops	In ES5, there is a use of **for** loop to iterate over elements.	ES6 introduced t to perform an ite the iterable objec

To learn more about the difference between ES5 and ES6, follow this link: ES5 v/s ES6

---

25) Define Modules in JavaScript.

Modules are the piece of JavaScript code written in a file. By using Modules, it is easy to maintain the code, debug the code, and reuse the code. Each module is a piece of code that gets executed once it is loaded.

---

26) What do you understand by the term Hoisting in JavaScript?

It is a JavaScript's default behavior, which is used to move all the declarations at the top of the scope before the execution of code. It can be applied to functions as well as on variables. It allows the JavaScript to use the component before its declaration. It does not apply to scripts that run in strict mode.

---

27) List the new Array methods introduced in ES6?

There are many array methods available in ES6, which are listed below:

- Array.of()
- Array.from()
- Array.prototype.copyWithin()
- Array.prototype.find()
- Array.prototype.findIndex()
- Array.prototype.entries()
- Array.prototype.keys()
- Array.prototype.values()
- Array.prototype.fill()

To learn more about the above array methods, follow this link: ES6 Array methods.

---

28) What are the new String methods introduced in ES6?

There are four string methods introduced in ES6 that are listed as follows:

- string.startsWith()
- string.endsWith()
- string.includes()
- string.repeat()

To learn more about the strings, follow this link: ES6 Strings.

---

29) Define Babel.

Babel is one of the popular transpilers of JavaScript. It is mainly used for converting the ES6 plus code into the backward-compatible version of JavaScript that can be run by previous JavaScript engines.

---

30) Define Webpack.

It is an open-source JavaScript module bundler that takes modules with dependencies. It allows us to run an environment that hosts Babel.