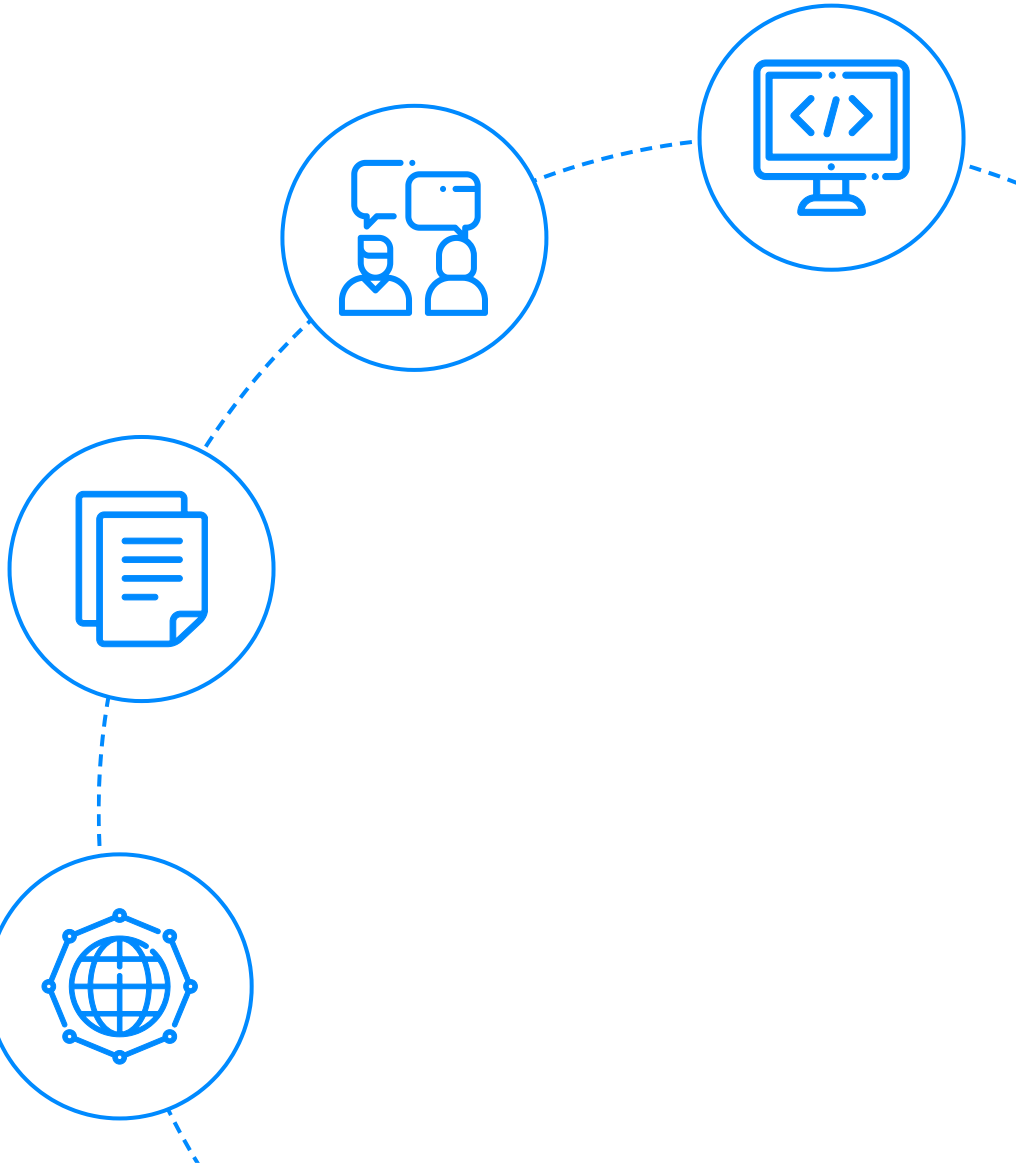
InterviewBit

ES6 Interview Questions



To view the live version of the page, click here.

Contents

ES6 Basic Interview Questions

ES6 Basic Interview Questions

(.....Continued)

21. What is Bubbling and Capturing?
22. What is the difference between for..of and for..in?
23. What is the reason behind adding Symbol to ES6?
24. What is Babel?
25. Name some array methods that were introduced in ES6.
26. Name some string functions introduced in ES6.
27. Compare the ES5 and ES6 codes for object initialization and parsing returned objects.
28. How do you use Destructuring Assignment to swap variables?
29. What is the result of the spread operator array shown below?

ES6 Interview Questions for Experienced

30. What is the Prototype Design Pattern?
31. What is a WeakMap in ES6? How is it different from a Map?
32. What is the advantage of using the arrow syntax for a constructor method?
33. What is a Temporal Dead Zone?
34. What is the difference between Set and WeakSet in ES6?
35. What are Proxy in ES6?
36. What is the difference between const and Object.freeze().
37. Why does the following not work as an IIFE (Immediately Invoked Function Expressions) ? What needs to be modified in order for it to be classified as an IIFE?
38. Explain Internationalization and Localization.

ES6 Interview Questions for Experienced (.....Continued)

39. What is Webpack?

Let's get Started

The name JavaScript was picked for marketing purposes when it first appeared in the programming realm. Java was becoming increasingly popular around the world at the time (both are anyways different). Javascript and its specification were later submitted to ECMA (European Computer Manufacturers Association) for standardisation.

It was later named as ECMAScript of ES. In June 1997, the first edition was released. ES6 is the language's sixth edition, which was eventually renamed ECMAScript 2015. Many new features were added in this release, including class, modules, iterators, for/of loops, arrow functions, typed arrays, promises, reflection, and more.

ES6 was launched in June 2016. The eighth edition was launched in June 2017 and added features such as concurrency and atomics, as well as syntactic integration with promises (async/await).

ES6 Basic Interview Questions

1. Mention some popular features of ES6.

Some of the common ES6 features are:

- Supports constants/immutable variables.
- Block scope support for all variables, constants and functions.
- Introduction to arrow functions
- Handling extended parameter
- Default parameters
- Extended and template literals
- De-structuring assignment
- Promises
- Classes
- Modules
- Collections
- Supports Map/Set & Weak-map/Weak-Set
- Localization, meta-programming, internationalization

2. What are the object oriented features supported in ES6.

The object-oriented features supported in ES6 are:

- **Classes**: We can create classes in ES6. The class function essentially builds a template from which we may later create objects. When a new instance of the class is created, the constructor method is invoked.
- **Methods**: Static methods can also be found in classes. A static method, unlike an object, is a function that is bound to the class. A static method can't be called from a class instance.
  Let's take a look at getters and setters for a moment. **Encapsulation** is a fundamental notion in OOP. Data (object properties) should not be directly accessed or updated from outside the object, which is a crucial aspect of encapsulation. A getter (access) or a setter (modify) are particular methods we define in our class to access or edit a property.
- **Inheritance**: It is also possible for classes to inherit from one another. The parent is the class that is being inherited from, and the child is the class that is inheriting from the parent.

3. Give a thorough comparison between ES5 and ES6.

ES5	ES6
ES5 is the fifth edition of the ECMAScript which was introduced in 2009.	ES6 is the sixth edition of the ECMAScript which was introduced in 2015.
Primitive data types that are **string, boolean, number, null,** and **undefined** are supported by ES5.	There are a few additions to the JavaScript data types in ES6. For supporting unique values, a new primitive data type **'symbol'** was introduced.
In ES5, we could define the variables by using the **var** keyword only.	In ES6, in addition to var, there are two new methods to define variables: **let** and **const**.
Both **function** and **return** keywords are used in order to define a function in ES5.	An arrow function is a newly added feature in ES6 in which we don't require the **function** keyword in order to define the function.
In ES5, a **for** loop is used to iterate over elements.	ES6 introduced the idea of **for...of** loop in order to iterate over the values of the iterable objects.

4. What is the difference between let and const? What distinguishes both from var?

When declaring any variable in JavaScript, we used the var keyword. Var is a function scoped keyword. Within a function, we can access the variable. When we need to create a new scope, we wrap the code in a function.

Both let and const have block scope. If you use these keywords to declare a variable, it will only exist within the innermost block that surrounds them. If you declare a variable with let inside a block (for example, if a condition or a for-loop), it can only be accessed within that block.

The variables declared with the let keyword are mutable, which means that their values can be changed. It's akin to the var keyword, but with the added benefit of block scoping. The variables declared with the const keyword are block-scoped and immutable. When variables are declared with the const keyword, their value cannot be modified or reassigned.

5. Discuss the arrow function.

In ES6, arrow functions are introduced. The shorthand syntax for writing ES6 functions is arrow functions. The arrow function's definition consists of parameters, followed by an arrow (=>), and the function's body.

The 'fat arrow' function is another name for the Arrow function. We won't be able to employ them as constructors.

```
 const function_name = (arg_1, arg_2, arg_3, ...) => {
    //body of the function
}
```

Few things to note:

- It reduces the size of the code.
- For a single-line function, the return statement is optional.
- Bind the context lexically.
- For a single-line statement, functional braces are not required.
- Doesn’t work with ***new***

6. When should one not use arrow functions?

One should not use arrow functions in the following cases:

- **Function Hoisting, Named Functions:**

As arrow functions are anonymous, we cannot use them when we want function hoisting or when we want to use named functions.

- **Object methods:**

```
 var a = {
  b: 7,
  func: () => {
    this.b--;
  }
}
```

The value of b does not drop when you call a.func. It's because this isn't bound to anything and will inherit the value from its parent scope.

- **Callback functions with dynamic context:**

```
 var btn = document.getElementById('clickMe');
btn.addEventListener('click', () => {
  this.classList.toggle('on');
});
```

We'd get a TypeError if we clicked the button. This is due to the fact that this is not bound to the button, but rather to its parent scope.

- **this/arguments:**

Since arrow functions don’t have this/arguments of their own and they depend on their outer context, we cannot use them in cases where we need to use this/arguments in a function.

7. What is the generator function?

This is a newly introduced feature in ES6. The Generator function returns an object after generating several values over time. We can iterate over this object and extract values from the function one by one. A generator function returns an iterable object when called. In ES6, we use the * sign for a generator function along with the new ‘yield' keyword.

```
 function *Numbers() {
    let num = 1;
    while(true) {
        yield num++;
    }
}

var gen = Numbers();

// Loop to print the first
// 5 Generated numbers
for (var i = 0; i < 5; i++) {

    // Generate the next number
    document.write(gen.next().value);

    // New Line
    document.write("<br>");
}
```

Output:

```
1
2
3
4
5
```

The yielded value becomes the next value in the sequence each time yield is invoked. Also, generators compute their output values on demand, allowing them to efficiently represent expensive to compute sequences or even infinite sequences.

8. What is the “spread” operator in ES6?

The list of parameters is obtained using the spread operator. Three dots (...) are used to represent it. The spread operator divides an iterable (such as an array or a string) into individual elements. It's mostly used in JavaScript to make shallow copies of JS. It improves the readability of your code by making it more concise.

The spread operator can be used to join two arrays together or to concatenate them.

```
let arr1 = [4, 5, 6];

let arr2 = [1, 2, 3, ...arr1, 7, 8, 9, 10];

console.log(arr2);
```

Output:

```
[ 1 2 3 4 5 6 7 8 9 10 ]
```

9. Explain Destructuring in ES6.

Destructuring was introduced in ES6 as a way to extract data from arrays and objects into a single variable. It is possible to extract smaller fragments from objects and arrays using this method. The following is an example.

```
let greeting =['Good','Morning'];
let [g1, g2] = greeting;
console.log (g1, g2);
```

Output:

```
Good Morning
```

10. What are Promises in ES6?

Asynchronous programming is a concept found in JavaScript. The processes are run separately from the main thread in asynchronous programming. Promises are the most convenient approach to deal with asynchronous programming in ES6. Depending on the outcome of the procedure, a promise can be refused or resolved. Callbacks were used to cope with asynchronous programming before promises were introduced in ES6.

However, it caused the problem of callback hell, which was addressed with the introduction of promises.

(A callback is a function that is performed after another function has completed. When working with events in JavaScript, callback is very useful. As an argument to another function, we pass a function into another function.

When we use callbacks in our web applications, it's common for them to get nested. Excessive callback usage clogs up your web application and leads to callback hell.)

11. Explain the Rest parameter in ES6.

It's a new feature in ES6 that enhances the ability to manage arguments. Indefinite arguments can be represented as an array using rest parameters. We can invoke a function with any number of parameters by utilizing the rest parameter.

```
function display(...args) {
   let ans = 0;
   for (let i of args) {
       ans *= i;
   }
   console.log("Product = "+ans);
 }

 display(4, 2, 3);
```

Output:

```
Product = 24
```

12. Discuss the template literals in ES6.

Template literals are a brand-new feature in ES6. It makes producing multiline strings and performing string interpolation simple.

Template literals, also known as string literals, allow for embedded expressions.

Template literals were referred to as template strings prior to ES6. The backtick (``) character is used to enclose template literals. The dollar sign and curly brackets (${expression}) are used to denote placeholders in template literals. If we need to use an expression within the backticks, we can put it in the (${expression}) variable.

```
let s1 = "Good";

let s2 = "Day";

let s = `${s1} ${s2}`;
console.log(s);
```

Output:

```
Good Day
```

13. Why should one use ES6 classes?

Developers have discovered that ES6 classes are really handy. The following are some of the most common applications of ES6 classes:

- The syntax of ES6 classes is simpler and less prone to errors.
- When it comes to building up inheritance hierarchies, ES6 is the ideal option because it combines new and old syntax, reducing errors and simplifying the process.
- ES6 classes prevent developers from making mistakes when using a new operator. If this proves to be an invalid object for the constructor, classes eliminate this issue by having the constructor throw an exception.
- Classes can also be used to call a method from the prototype's version. With the new ES6 syntax, this version is significantly easier to use than previous versions.

14. How can you create a class in ES6?

The keyword class is used to create a class in ES6. We can use class expressions or class declarations to include classes in our code. Only functions and constructors are allowed in a class definition. These components are collectively referred to as the class's data members.

Constructors in classes are responsible for allocating memory to the class's objects. A class's functions are in charge of performing actions on the objects.

Syntax: In ES5

```
 var varName = new className {
}
```

Syntax: In ES6 (Using class keyword)

```
 class className{
}
```

15. What is a class expression?

In ES6, one way to define a class is to use the Class expression. Class expressions, like function expressions, can be named or unnamed. If the class is named, the name is unique to the class body. Prototype-based inheritance is used in JavaScript classes.

```
 var Product = class {
    constructor (num1, num2) {
    this.num1 = num1;
    this.num2 = num2;
    }
    multiplication() {
    return this.num1 * this.num2;
    }
}
console.log(new Product(5,8).multiplication());
// expected output: 40
```

The syntax of a class expression is similar to that of a class statement (declaration). Class expressions, on the other hand, allow you to omit the class name (“binding identifier”), which is not possible with class statements. Additionally, unlike class declarations, class expressions allow you to redefine/re-declare classes without causing any type errors. It is not required to use the constructor property. The type of classes created with this keyword will always be "function."

16. What do you understand about default parameters?

If no value or undefined is passed, we can use the default parameters to set default values for named parameters.

```
 var display = (x , y = 2) => {
    console.log(x + " " + y);
}
display(1);
```

Output:

```
 1 2
```

17. What do you understand about IIFE (Immediately Invoked Function Expressions)?

IIFE is a JavaScript function that starts running as soon as it is defined. The Self-Executing Anonymous Function is another name for it. It is divided into two major sections, which are as follows:

- The first part is a lexical scope (static scope) anonymous function that is enclosed by the Grouping operator ().
- The IIFE, which is used by JavaScript, is created in the second part. The function will be directly interpreted by the engine.

```
 (func_()
{
   console.log("Good Day");
})();
```

Output:

```
 Good Day
```

18. What are the states of Promises in ES6?

Promises mainly possess three states as follows:

- **Pending:** This refers to the initial state of every promise. It indicates that the result has not yet been computed.
- **Fulfilled:** It refers to the completion of a task.
- **Rejected:** It indicates the failure that arises during computation.

The promise will be immutable once it has been fulfilled or rejected. A rejected function and a resolve function are the two arguments passed into the Promise() constructor. It returns either the first or second parameter, depending on the asynchronous operation.

19. What is Export Default and Named Export in ES6?

With the help of the import statement, the export statement comes into picture when one needs to export functions, objects, and variables to other JavaScript modules. There are two methods for exporting:

- **Named Export:** Named exports are useful when one has to export multiple values. The name of the imported module must match that of the exported module.

Example:

```
 //file rectangle.js
function perimeter(x, y) {
  return 2 * (x + y);
}
function area(x, y) {
  return x * y;
}
export { perimeter, area };


//while importing the functions in test.js
import { perimeter, area } from './rectangle;
console.log(perimeter(4, 6)) //20
console.log(area(4, 6)) //24
```

Output:

```
20
24
```

- **Default Export:** There is only one default export per module when it comes to default exports. A function, a class, an object, or anything else can be used as a default export. In default export, the naming of imports is fully autonomous, and we can choose any name we like.

Example:

```
 // file module.js
var a = 6;
export default a;

// test.js
// while importing a in test.js
import b from './module';
console.log(b);
// output will be 6
```

Output:

```
6
```

- **Using Named and Default Exports at the same time**: In the same file, you can utilise both Named and Default exports. It means they'll both be imported into the same document.

```
 //index.js
var a = 3;
const b = 8;
function show() {
   return "This is a default export."
}
function product(a , b) {
  return a * b;
}
export { show as default, a, b, product };
```

```
//test.js file
import any_other_name, { a, b, product} from './index.js';
console.log(any_other_name()); //This is a default export.
console.log(a); //3
```

Output:

```
This is a default export.
3
```

20. Which keyword can be used to deploy inheritance in ES6?

The extend keyword is used to implement inheritance in the ES6 language. There was no idea of classes in prior versions of Javascript, but with the release of ES6, Pure Object Oriented elements were added to the language.

```
 class Classroom {
    constructor(students) {
        this.students = students;
    }
    room() {
        console.log('This class has  ' + this.students + ' students');
    }
}

class sectionA extends Classroom {
    constructor(students) {
        super(students);
    }
    sec() {
        console.log('section A');
    }
}

let secA = new sectionA(40);

secA.room();
secA.sec();
```

21. What is Bubbling and Capturing?

When an event occurs on the DOM, it does not take place completely on one element. The event bubbles up or goes to its parent, grandparents, and grandparent's parent until it reaches the window in the Bubbling Phase, whereas the event starts out from window down to the element that prompted the event or the event target in the Capturing Phase.

There are three stages of event propagation:

- **Capturing Phase** - the event begins with the window and progresses through each element until it reaches the target element.
- **Target Phase** - The event has arrived at the target element.
- **Bubbling Phase** - The event bubbles up from the target element and then up every element until it reaches the window.

22. What is the difference between for..of and for..in?

- **for in**: runs over an object's enumerable property names.
- **for of**: (new in ES6) takes an object-specific iterator and loops through the data it generates.

Both the **for..of** and **for..in** commands iterate over lists, but the results they return are different: **for..in** returns a list of keys on the object being iterated, whereas **for..of** returns a list of values of the object's numeric attributes.

```
let arr = [3, 4, 5];

for (let i in arr) {
   console.log(i); // "0", "1", "2",
}

for (let i of arr) {
   console.log(i); // "3", "4", "5"
}
```

23. What is the reason behind adding Symbol to ES6?

Symbols are a new type of object that can be used as distinct property names in objects. Using Symbols instead of strings allows separate modules to create properties that are not mutually exclusive. Symbols can also be kept private, preventing anyone who does not have direct access to the Symbol from accessing its properties.

Symbols are a brand-new kind of primitive. Symbols, like numbers, strings, and booleans, have a function that can be used to produce them. Symbols, unlike the other primitives, do not have a literal syntax (similar to how strings have '') and can only be created using the Symbol constructor:

```
let symbol = Symbol();
```

In truth, Symbols are only a little different means of attaching properties to an object; the well-known Symbols could easily be provided as standard methods, just like Object.prototype.has Own Property which appears in anything that inherits from Object.

24. What is Babel?

Babel is an open-source JavaScript transpiler that converts ECMAScript 2015+ (ES6+) code into a backwards compatible version of JavaScript that can be run by previous JavaScript engines. Babel is a popular tool for exploiting the JavaScript programming language's latest capabilities.

Babel plugins are used to convert syntax that isn't widely supported into a version that is backwards compatible. Arrow functions, for example, which are defined in ES6, are translated to ordinary function declarations. It's also possible to translate non-standard JavaScript syntax, such as JSX. Babel may automatically inject core-js polyfills for support capabilities that aren't available in JavaScript environments. Static methods like Array.from and built-ins like Promise, for example, are only accessible in ES6+, but they can be utilised in previous contexts by using core-js.

25. Name some array methods that were introduced in ES6.

Methods	Description
Array.from()	It will convert iterable values and array-like values into arrays.
Array.of()	It will create a new array instance from a variable number of arguments no matter what the number or the type of arguments are.
Array.prototype.copyWithin()	It will copy the portion of an array to a different place within the same array.
Array.prototype.find()	It will find an element in an array, based on certain parameters that are passed into this method.
Array.prototype.findIndex()	It will return the index of the first element of the given array that fulfills the given condition.
Array.prototype.entries()	It will return an array iterator object that can be used while looping through the keys and values of arrays.
Array.prototype.keys()	It will return an array iterator object as well as the keys of the array.
Array.prototype.values()	It will provide the value of each key

26. Name some string functions introduced in ES6.

Methods	Description
startsWith	It determines if a string begins with the characters of a given string.
endsWith	It determines if a string ends with the characters of a given string.
includes	It will return true if the given argument is present in the string.
repeat	It creates and returns a new string which contains the given number of copies of the string on which this method was called, concatenated together.

27. Compare the ES5 and ES6 codes for object initialization and parsing returned objects.

Object initialization: Variables with the same name are frequently used to create object properties. Consider the following scenario:

```
 // ES5 code
var
  x = 1, y = 2, z = 3;
  ob = {
    x : a,
    y : b,
    z : z
  };

// ob.x = 1, ob.y = 2, ob.z = 3
```

In ES6, there's no need for tedious repetition!

```
 // ES6 code
const
  x = 1, y = 2, z = 3;
  ob = {
    x
    y
    z
  };

// ob.x = 1, ob.y = 2, ob.z = 3
```

Parsing returned objects: Only one value can be returned by a function, but that value could be an object with hundreds of properties and/or methods. In ES5, you must first get the returned object and then extract values from it. Consider the following scenario:

```
 // ES5 code
var
  ob = getObject(),
  a = ob.a,
  b = ob.b,
  c = ob.c;
```

This is made easier by ES6 destructuring, which eliminates the need to keep the object as a variable:

```
 // ES6 code
const { a , b , c } = getObject();
```

28. How do you use Destructuring Assignment to swap variables?

```
var a = 1, b = 2;

[a, b] = [b, a];
console.log(a); // 2
console.log(b); // 1
```

29. What is the result of the spread operator array shown below?

```
[...'apple']
```

Output: ['a', 'p', 'p', 'l', 'e']

Explanation: A string is an iterable type, and in an array, the spread operator transfers each character of an iterable to one element. As a result, each character in a string becomes an Array element.

ES6 Interview Questions for Experienced

30. What is the Prototype Design Pattern?

The Prototype Pattern creates new objects, but instead of returning uninitialized objects, it returns objects with values copied from a prototype - or sample - object. The Properties pattern is another name for the Prototype pattern.

The initialization of business objects with values that match the database's default settings is an example of where the Prototype pattern comes in handy. The default values from the prototype object are replicated into a newly generated business object.

The Prototype pattern is rarely used in traditional languages, but JavaScript, as a prototypal language, employs it in the creation of new objects and prototypes.

31. What is a WeakMap in ES6? How is it different from a Map?

The WeakMap is a collection of key/value pairs, just like the Map. The keys of WeakMap, on the other hand, must be objects, whereas the values can be anything. The object references in the keys are held weakly, which means that if there is no other reference to the object, it will be eligible for garbage collection. WeakMap, but not Map, permits the garbage collector to complete its duty. The array of keys would preserve references to key objects in manually constructed maps, prohibiting them from being garbage collected. References to key objects in native WeakMaps are held "weakly," which means they do not hinder garbage collection if there is no other reference to the object. The Map API and the WeakMap API are the same.

WeakMap keys, on the other hand, are not enumerable, unlike Map objects. There are also no methods that return a list of keys. If they were, the list would be non-deterministic because it would be dependent on the state of garbage collection. A Map should be used if we need a list of keys.

32. What is the advantage of using the arrow syntax for a constructor method?

The main benefit of utilising an arrow function as a method within a constructor is that the value of ***this*** is set at the moment of function generation and cannot be changed later. As a result, whenever the constructor is used to create a new object, ***this*** refers to that object.

```
 const Shape = function(shapeName) {
  this.shapeName = shapeName;
  this.showName1 = function() { console.log(this.shapeName); };
  this.showName2 = () => { console.log(this.shapeName); };
};

const circle = new Shape('Circle');
const square = new Shape('Square');

circle.showName1(); // Circle
circle.showName2(); // Square

// The regular function can have its 'this' value changed, but the arrow function cannc
circle.showName1.call(square); // Square (because "this" is now the square object)
circle.showName2.call(square); // Circle

circle.showName1.apply(square); // Square (because 'this' is now the square object)
circle.showName2.apply(square); // Circle

circle.showName1.bind(square)(); // Square (because 'this' is now the square object)
circle.showName2.bind(square)(); // Circle

var showNameFromPic1 = circle.showName1;
sayNameFromPic1(); // undefined (because 'this' is now the pic object)

var showNameFromPic2 = circle.showName2;
showNameFromPic2(); // Circle
```

The major point here is that for a normal function, ***this*** can be modified, but for an arrow function, the context is always the same. You won't have to worry about the context changing if you pass your arrow function around to other areas of your application.

33. What is a Temporal Dead Zone?

Variable Hoisting does not apply to let bindings in ES6, so let declarations do not rise to the top of the current execution context. A ReferenceError is thrown if the variable in the block is referenced before it is initialized (unlike a variable declared with var, which will just possess the undefined value). From the beginning of the block until the initialization is performed, the variable is in a "temporal dead zone."

```
console.log(a); // undefined
console.log(b); // causes ReferenceError: aLet is not defined
var a = 1;
let b = 2;
```

34. What is the difference between Set and WeakSet in ES6?

Set: By using the Set() class, users can define an array-like heterogeneous iterable object, which will consist of distinct values. The elements should not just be distinct by values but also by types. i.e. "2" and 2 will be considered as different.

```
var set1= new Set([0, 1, 2]);

set1.add(3); // 0, 1, 2, 3
set1.add(2); // 0, 1, 2, 3
set1.add({x:1, y:2}); // 0, 1, 2, {x:1, y:2}
set1.add("Good"); // 0, 1, 2, {x:1, y:2}, 'Good'

set1.has("Hello"); // false
set1.has("Good");  // true
set1.delete("Good"); // 'Good' deleted
set1.has("Good"); // false

set1.size; // 4
set1.clear(); // Set Cleared
```

WeakSet(): A WeakSet() is a collection that is similar to a Set because it retains unique values; but it can only hold Objects. If an object in your WeakSet has no other reference variables left, it will be removed automatically.

```
 var weakSet1 = new WeakSet([{x:1}]);
var ob1 = {o:1};
var ob2 = {o:2};

weakSet1.has(ob1); //false
weakSet1.add(ob1);
weakSet1.add(ob2);
weakSet1.has(ob2); // true
delete ob1; // you can't delete objects in this way. Use scope to execute this.
myWeakSet.has(ob1); // false, because you deleted ob1, so WeakSet releases it automatic
myWeakSet.delete(ob2); // ob2 deleted from the set
myWeakSet.add(1); // ERROR, no primitive value
```

Set	WeakSet
A set can contain all types of values.	A weakSet can only contain objects.
Use .size to find the number of elements.	Use .length to find the number of elements.
.forEach() is available for iteration.	.forEach() is not available for iteration.
Nothing is auto-destroyed.	An element object will be auto released to the garbage collector if it has no other reference left.

35. What are Proxy in ES6?

The proxy objects are used to customize behaviour for basic operations like property lookup, assignment, enumeration, function invocation, etc.

For basic actions, the Proxy object is used to create custom behaviour (e.g. property lookup, assignment, enumeration, function invocation, etc).

We need to define three crucial terms:

- **handler** — a placeholder object that holds the trap(s)
- **traps** — the method(s) that let you access a property.
- **target** — the virtualized object by the proxy

Example:

```
 const handle = {
  get: function(ob, prp) {
    return prp in ob ?
      ob[prp] :
      37;
  }
};

const x = new Proxy({}, handle);
x.a = 2;
x.b = undefined;

console.log(x.a, x.b);
//  2, undefined

console.log('c' in x, x.c);
//  false, 37
```

Proxies have a wide range of real-world applications.

- validation
- correction in value
- extensions for property lookup
- property accesses are being tracked
- references that can be revoked
- implementation of the DOM in javascript

36. What is the difference between const and Object.freeze().

Const is a property that applies to bindings ("variables"). It creates an immutable binding, which means you can't change its value.

```
 const a = {
    b: "apple"
};
let c = {
    d: "mango"
};
a = c; // ERROR "a" is read-only
```

Object.freeze() is a function that works with values, primarily object values. It makes an object immutable, meaning that its properties cannot be changed.

freeze() returns the same object that was supplied to it as a parameter. It does not make a frozen copy. If the parameter to this method is not an object (a primitive) in ES5, a TypeError will occur. A non-object argument in ES6 will be treated as if it were a frozen regular object and will be returned.

```
 let a = {
    b: "apple"
};
let c = {
    d: "mango"
};
Object.freeze(a);
a.b = "kiwi"; //TypeError: Cannot assign to read only property 'name' of object
console.log(a);
```

37. Why does the following not work as an IIFE (Immediately Invoked Function Expressions) ? What needs to be modified in order for it to be classified as an IIFE?

```
function f(){ }();
```

The JavaScript parser interprets the function f(){ }(); as function f(){ } and (); where the former is a function declaration while the latter (a pair of brackets) is an attempt to execute a function without specifying a name, resulting in Uncaught SyntaxError: Unexpected token).

(function f(){ })() and (function f(){ }()) are two ways to fix it that involve adding more brackets. These functions are not available in the global scope, and you can even omit their names if you don't need to refer to them within the body.

You can also use the void operator: void function f(){ }(); .However, there is a drawback in this strategy. Because the evaluation of a given expression is always undefined, you can't use your IIFE function if it returns anything. Consider the following scenario:

```
 // Don't add JS syntax to this code block to prevent Prettier from formatting it.
const f = void
function b() {
    return 'f';
}();

console.log(f); // undefined
```

38. Explain Internationalization and Localization.

These are JavaScript standard APIs that assist with operations such as collation, number formatting, currency formatting, and date and time formatting.

- **Collation**: It is a method for searching and sorting strings within a collection. It has a locale argument and is Unicode-aware.
- **Number Formatting**: Localized separators and digit grouping can be used to format numbers. Style formatting, numeral system, percent, and precision are among the other items.
- **Currency formatting**: Currency symbols, localized separators, and digit grouping are the most common ways to format numbers.
- **Date and time formatting**: Localized separators and ordering are used for formatting. The format can be short or long, and other characteristics such as location and time zone can be included.

39. What is Webpack?

Webpack is a tool for bundling javascript files for usage in browsers. Webpack analyses the application and generates the bundles by creating a dependency graph that maps each module of the project required. It enables you to execute the environment that was hosted by Babel. A web pack has the advantage of combining numerous modules and packs into a single JavaScript file. It includes a dev server, which aids with code updates and asset management.

Write your code:

folder_name/index.js

```
 import bar from './func.js';
func();
```

folder_name/func.js

```
 export default function func() {
  // body of the function
}
```

Bundle it:

Either provide with custom webpack.config.js or without config:

```
 const path = require('path');

module.exports = {
  entry: './folder_name/index.js',
  output: {
    path: path.resolve(__dirname, 'dict'),
    filename: 'bundle.js',
  },
};
```

page.html

```
 <!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    ...
  </head>
  <body>
    ...
    <script src="dict/bundle.js"></script>
  </body>
</html>
```

You, then, need to run the webpack on the command-line in order to create bundle.js.

Links to More Interview Questions

C Interview Questions	Php Interview Questions	C Sharp Interview Questions
Web Api Interview Questions	Hibernate Interview Questions	Node Js Interview Questions
Cpp Interview Questions	Oops Interview Questions	Devops Interview Questions
Machine Learning Interview Questions	Docker Interview Questions	Mysql Interview Questions
Css Interview Questions	Laravel Interview Questions	Asp Net Interview Questions
Django Interview Questions	Dot Net Interview Questions	Kubernetes Interview Questions
Operating System Interview Questions	React Native Interview Questions	Aws Interview Questions
Git Interview Questions	Java 8 Interview Questions	Mongodb Interview Questions
Dbms Interview Questions	Spring Boot Interview Questions	Power Bi Interview Questions
Pl Sql Interview Questions	Tableau Interview Questions	Linux Interview Questions
Ansible Interview Questions	Java Interview Questions	Jenkins Interview Questions