22 Important JavaScript/ES6/ES2015 Interview Questions

JavaScript 142

> ES6 refers to version 6 of the ECMA Script programming language. ECMA Script is the standardized name for JavaScript, and version 6 is the next version after version 5, which was released in 2011. It is a major enhancement to the JavaScript language, and adds many more features intended to make large-scale software development easier. Both ES6 and ES2015 names used for the version of JavaScript that introduces arrow functions, template strings, and Promises.

Q1: Could you explain the difference between ES5 and ES6

Mid JavaScript 142

Answer

- **ECMAScript 5 (ES5)**: The 5th edition of ECMAScript, standardized in 2009. This standard has been implemented fairly completely in all modern browsers
- **ECMAScript 6 (ES6)/ ECMAScript 2015 (ES2015)**: The 6th edition of ECMAScript, standardized in 2015. This standard has been partially implemented in most modern browsers.

Here are some key differences between ES5 and ES6:

- **Arrow functions** & **string interpolation**:
  Consider:

```
const greetings = (name) => {
      return `hello ${name}`;
}
```

and even:

```
const greetings = name => `hello ${name}`;
```

- **Const**.
  Const works like a constant in other languages in many ways but there are some caveats. Const stands for 'constant reference' to a value. So with const, you can actually mutate the properties of an object being referenced by the variable. You just can't change the reference itself.

```
const NAMES = [];
NAMES.push("Jim");
console.log(NAMES.length === 1); // true
NAMES = ["Steve", "John"]; // error
```

- **Block-scoped variables**.
  The new ES6 keyword `let` allows developers to scope variables at the block level. `Let` doesn't hoist in the same way `var` does.
- **Default parameter values** Default parameters allow us to initialize functions with default values. A default is used when an argument is either omitted or undefined — meaning null is a valid value.

```
// Basic syntax
function multiply (a, b = 2) {
```

```
    return a * b;
}
multiply(5); // 10
```

- **Class Definition and Inheritance**
  ES6 introduces language support for classes (`class` keyword), constructors (`constructor` keyword), and the `extend` keyword for inheritance.

- **for-of operator**
  The for...of statement creates a loop iterating over iterable objects.

- **Spread Operator** For objects merging

```
const obj1 = { a: 1, b: 2 }
const obj2 = { a: 2, c: 3, d: 4}
const obj3 = {...obj1, ...obj2}
```

- **Promises**
  Promises provide a mechanism to handle the results and errors from asynchronous operations. You can accomplish the same thing with callbacks, but promises provide improved readability via method chaining and succinct error handling.

```
const isGreater = (a, b) => {
  return new Promise ((resolve, reject) => {
    if(a > b) {
      resolve(true)
    } else {
      reject(false)
    }
    })
}
isGreater(1, 2)
  .then(result => {
    console.log('greater')
  })
 .catch(result => {
    console.log('smaller')
 })
```

- **Modules exporting & importing** Consider module exporting:

```
const myModule = { x: 1, y: () => { console.log('This is ES5') }}
export default myModule;
```

and importing:

```
import myModule from './myModule';
```

Having Tech Interview? Check 👉 142 JavaScript Interview Questions

Source: FullStack.Cafe

Q2: Explain the difference between `Object.freeze()` vs `const`

Mid JavaScript 142

Answer

`const` and `Object.freeze` are two completely different things.

- `const` applies to **bindings** ("variables"). It creates an immutable binding, i.e. you cannot assign a new value to the binding.

```
const person = {
    name: "Leonardo"
};
let animal = {
    species: "snake"
};
person = animal; // ERROR "person" is read-only
```

- `Object.freeze` works on **values**, and more specifically, *object values*. It makes an object immutable, i.e. you cannot change its properties.

```
let person = {
    name: "Leonardo"
};
let animal = {
    species: "snake"
};
Object.freeze(person);
person.name = "Lima"; //TypeError: Cannot assign to read only property 'name' of object
console.log(person);
```

Having Tech Interview? Check 👉 142 JavaScript Interview Questions

Source: stackoverflow.com

Q3: What are the benefits of using *spread syntax* in ES6 and how is it different from *rest syntax*?

Mid JavaScript 142

Answer

ES6's spread syntax is very useful when coding in a functional paradigm as we can easily create copies of arrays or objects without resorting to `Object.create`, `slice`, or a library function. This language feature is used often in Redux and rx.js projects.

```
function putDookieInAnyArray(arr) {
  return [...arr, 'dookie'];
}
```

```
const result = putDookieInAnyArray(['I', 'really', "don't", 'like']); // ["I", "really",

const person = {
  name: 'Todd',
  age: 29,
};

const copyOfTodd = { ...person };
```

ES6's rest syntax offers a shorthand for including an arbitrary number of arguments to be passed to a function. It is like an inverse of the spread syntax, taking data and stuffing it into an array rather than unpacking an array of data, and it works in function arguments, as well as in array and object destructuring assignments.

```
function addFiveToABunchOfNumbers(...numbers) {
  return numbers.map(x => x + 5);
}

const result = addFiveToABunchOfNumbers(4, 5, 6, 7, 8, 9, 10); // [9, 10, 11, 12, 13, 1

const [a, b, ...rest] = [1, 2, 3, 4]; // a: 1, b: 2, rest: [3, 4]

const { e, f, ...others } = {
  e: 1,
  f: 2,
  g: 3,
  h: 4,
}; // e: 1, f: 2, others: { g: 3, h: 4 }
```

Having Tech Interview? Check 👉 142 JavaScript Interview Questions

Source: github.com/yangshun

Mid JavaScript 142

Q4: What are the differences between ES6 class and ES5 function constructors?

Answer

Let's first look at example of each:

```
// ES5 Function Constructor
function Person(name) {
  this.name = name;
}

// ES6 Class
class Person {
  constructor(name) {
    this.name = name;
  }
}
```

For simple constructors, they look pretty similar.

The main difference in the constructor comes when using inheritance. If we want to create a `Student` class that subclasses `Person` and add a `studentId` field, this is what we have to do in addition to the above.

```
// ES5 Function Constructor
function Student(name, studentId) {
  // Call constructor of superclass to initialize superclass-derived members.
  Person.call(this, name);

  // Initialize subclass's own members.
  this.studentId = studentId;
}

Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;

// ES6 Class
class Student extends Person {
  constructor(name, studentId) {
    super(name);
    this.studentId = studentId;
  }
}
```

It's much more verbose to use inheritance in ES5 and the ES6 version is easier to understand and remember.

Having Tech Interview? Check 👉 142 JavaScript Interview Questions

Source: github.com/yangshun

Q5: What is IIFEs (Immediately Invoked Function Expressions)?

Mid JavaScript 142

Answer

It's an Immediately-Invoked Function Expression, or IIFE for short. It executes immediately after it's created:

```
(function IIFE(){
    console.log( "Hello!" );
})();
// "Hello!"
```

This pattern is often used when trying to avoid polluting the global namespace, because all the variables used inside the IIFE (like in any other normal function) are not visible outside its scope.

Having Tech Interview? Check 👉 142 JavaScript Interview Questions

Source: stackoverflow.com

29 Flutter Interview Questions Mobile Devs Need To Know (ANSWERED for 2020)

Flutter 68

Q6: What is generator in JS?

Mid JavaScript 142

Answer

Generators are functions which can be exited and later re-entered. Their context (variable bindings) will be saved across re-entrances. Generator functions are written using the `function*` syntax. When called initially, generator functions do not execute any of their code, instead returning a type of iterator called a Generator. When a value is consumed by calling the generator's `next` method, the Generator function executes until it encounters the `yield` keyword.

The function can be called as many times as desired and returns a new Generator each time, however each Generator may only be iterated once.

```
function* makeRangeIterator(start = 0, end = Infinity, step = 1) {
    let iterationCount = 0;
    for (let i = start; i < end; i += step) {
        iterationCount++;
        yield i;
    }
    return iterationCount;
}
```

Having Tech Interview? Check 👉 142 JavaScript Interview Questions

Source: stackoverflow.com

Q7: What is the motivation for bringing `Symbol` to ES6?

Mid JavaScript 142

Answer

`Symbols` are a new, special kind of object that can be used as a unique property name in objects. Using `Symbol` instead of `string` 's allows different modules to create properties that don't conflict with one another. `Symbols` can also be made private, so that their properties can't be accessed by anyone who doesn't already have direct access to the `Symbol` .

`Symbols` are a new **primitive**. Just like the `number` , `string` , and `boolean` primitives, `Symbol` have a function which can be used to create them. Unlike the other primitives, `Symbols` do not have a literal syntax (e.g how `string` have `''`) - the only way to create them is with the `Symbol` constructor in the following way:

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

In reality, `Symbol` 's are just a slightly different way to attach properties to an object - you could easily provide the well-known `Symbols` as standard methods, just like `Object.prototype.hasOwnProperty` which appears in everything that inherits from `Object` .

Having Tech Interview? Check 👉 142 JavaScript Interview Questions

Source: stackoverflow.com

Q8: What is the preferred syntax for defining enums in JavaScript?

Mid JavaScript 142

Answer

Since 1.8.5 it's possible to seal and freeze the object, so define the above as:

```
var DaysEnum = Object.freeze({
    "monday": 1,
    "tuesday": 2,
    "wednesday": 3,
    ...
})
```

or

```
var DaysEnum = {
    "monday": 1,
    "tuesday": 2,
    "wednesday": 3,
    ...
}
Object.freeze(DaysEnum)
```

and voila! JS enums.

However, this doesn't prevent you from assigning an undesired value to a variable, which is often the main goal of enums:

```
let day = DaysEnum.tuesday
day = 298832342 // goes through without any errors
```

Having Tech Interview? Check 👉 142 JavaScript Interview Questions

Source: stackoverflow.com

Ionic 29 | Golang 49 | ASP.NET Web API 33 | MySQL 55

Reactive Programming 12 | Git 36 | GraphQL 25 | PWA 22

T-SQL 36 | Design Patterns 45

Q9: What's the difference between `.call` and `.apply` ?

Mid | JavaScript 142

Answer

Both `.call` and `.apply` are used to invoke functions and the first parameter will be used as the value of `this` within the function. However, `.call` takes in comma-separated arguments as the next arguments while `.apply` takes in an array of arguments as the next argument. An easy way to remember this is C for `call` and comma-separated and A for `apply` and an array of arguments.

```
function add(a, b) {
  return a + b;
}

console.log(add.call(null, 1, 2)); // 3
console.log(add.apply(null, [1, 2])); // 3
```

Having Tech Interview? Check 👉 142 JavaScript Interview Questions

Source: github.com/yangshun

35 LINQ Interview Questions and Answers in 2019

λ LINQ 38

Q10: When should I use Arrow Functions in ES6?

Mid | JavaScript 142

Answer

I'm now using the following rule of thumb for functions in ES6 and beyond:

- Use `function` in the global scope and for Object.prototype properties.
- Use `class` for object constructors.
- Use `=>` everywhere else.

Why use arrow functions almost everywhere?

- **Scope safety**: When arrow functions are used consistently, everything is guaranteed to use the same thisObject as the root. If even a single standard function callback is mixed in with a bunch of arrow functions there's a chance the scope will become messed up.
- **Compactness**: Arrow functions are easier to read and write. (This may seem opinionated so I will give a few examples further on).
- **Clarity**: When almost everything is an arrow function, any regular function immediately sticks out for defining the scope. A developer can always look up the next-higher function statement to see what the thisObject is.

Having Tech Interview? Check 👉 142 JavaScript Interview Questions

Source: stackoverflow.com

Q11: Why should we use ES6 classes?

Mid JavaScript 142

Answer

Some reasons you might choose to use **Classes**:

- The syntax is simpler and less error-prone.
- It's **much** easier (and again, less error-prone) to set up inheritance hierarchies using the new syntax than with the old.
- `class` defends you from the common error of failing to use `new` with the constructor function (by having the constructor throw an exception if `this` isn't a valid object for the constructor).
- Calling the parent prototype's version of a method is much simpler with the new syntax than the old (`super.method()` instead of `ParentConstructor.prototype.method.call(this)` or `Object.getPrototypeOf(Object.getPrototypeOf(this)).method.call(this)`).

Consider:

```
// **ES5**
var Person = function(first, last) {
    if (!(this instanceof Person)) {
        throw new Error("Person is a constructor function, use new with it");
    }
    this.first = first;
    this.last = last;
};

Person.prototype.personMethod = function() {
    return "Result from personMethod: this.first = " + this.first + ", this.last = " + t
};

var Employee = function(first, last, position) {
    if (!(this instanceof Employee)) {
        throw new Error("Employee is a constructor function, use new with it");
    }
    Person.call(this, first, last);
    this.position = position;
};
```

```
Employee.prototype = Object.create(Person.prototype);
Employee.prototype.constructor = Employee;
Employee.prototype.personMethod = function() {
    var result = Person.prototype.personMethod.call(this);
    return result + ", this.position = " + this.position;
};
Employee.prototype.employeeMethod = function() {
    // ...
};
```

And the same with ES6 classes:

```
// ***ES2015+***
class Person {
    constructor(first, last) {
        this.first = first;
        this.last = last;
    }

    personMethod() {
        // ...
    }
}

class Employee extends Person {
    constructor(first, last, position) {
        super(first, last);
        this.position = position;
    }

    employeeMethod() {
        // ...
    }
}
```

Having Tech Interview? Check 👉 142 JavaScript Interview Questions

Source: stackoverflow.com

Q12: Could you compare usage of Module Pattern vs Constructor/Prototype pattern?

Senior JavaScript 142

Answer

Join **FullStack.Cafe** to open this Answer. It's Free!

Get Free Access To Answer

Q13: Explain the Prototype Design Pattern

Senior JavaScript 142

Answer

Join **FullStack.Cafe** to open this Answer. It's Free!

Get Free Access To Answer

17 Common Agile Interview Questions For Experienced Software Developers (CLARIFIED)

Agile & Scrum 51

Q14: Explain why the following doesn't work as an IIFE. What needs to be changed to properly make it an IIFE?

Senior JavaScript 142

Answer

Join **FullStack.Cafe** to open this Answer. It's Free!

Get Free Access To Answer

Q15: What are the actual uses of ES6 WeakMap?

Senior JavaScript 142

Answer

Join **FullStack.Cafe** to open this Answer. It's Free!

Get Free Access To Answer

Q16: What is Hoisting in JavaScript?

Senior JavaScript 142

Answer

Join **FullStack.Cafe** to open this Answer. It's Free!

Get Free Access To Answer

Q17: What is the Temporal Dead Zone in ES6?

Senior JavaScript 142

Answer

Join **FullStack.Cafe** to open this Answer. It's Free!

Get Free Access To Answer

25 NoSQL Interview Questions (ANSWERED) You Must Know In 2020

MongoDB 83 NoSQL 16 SQL 56

Q18: When should you NOT use arrow functions in ES6? Name three or more cases.

Senior JavaScript 142

Answer

Join **FullStack.Cafe** to open this Answer. It's Free!

Get Free Access To Answer

Q19: Can you give an example of a curry function and why this syntax offers an advantage?

Expert JavaScript 142

Answer

Unlock **FullStack.Cafe Membership Access** to open All Answers...

Unlock All Answers

Q20: Compare `Async/Await` and `Generators` usage to achive same functionality

Expert JavaScript 142

Answer

Unlock **FullStack.Cafe Membership Access** to open All Answers...

Unlock All Answers

Q21: How to *deep-freeze* object in JavaScript?

Expert JavaScript 142

Answer

Unlock **FullStack.Cafe Membership Access** to open All Answers...

Unlock All Answers

Guest Voice: The Most Asked Go Interview Questions By ScienceSoft

Golang 49

Q22: What's the difference between ES6 `Map` and `WeakMap` ?

Expert JavaScript 142

Answer

Unlock **FullStack.Cafe Membership Access** to open All Answers...

Unlock All Answers

OOP 54 MySQL 55 Redux 30 Android 113

Spring 87 Java 147 Vue.js 41 Ruby on Rails 72

WCF 33 Kotlin 68

29 Advanced MySQL Interview Questions Developers Must Know Before Tech Interview

MySQL 55

MySQL has emerged as the most popular SQL-based RDBMS in 2019 with almost 40% of users opting for this platform. It is widely used as a stand-alone database solution as well as in combination with other solutions such as MongoDB and PostgreSQL. If yo...

27 Binary Tree Interview Questions (SOLVED with CODE) Devs Must Know

Binary Tree 26

Amongst different types of data structures are binary trees that come with more uses than most of the other types. The reason that binary trees are used more often than n-ary trees (for example in searching) is that n-ary trees are more complex, but ...

32 Advanced Data Structures Interview Questions (ANSWERED) To Smash Your Next Programming Interview

Data Structures 44

Most interviews these days are whiteboard interviews demonstrating your knowledge of DSA. Unless you're interviewing for a top company, you can expect the questions to be fairly standard. The reason why questions about Data Structures are asked, has ...

40 Coding Challenges (SOLVED with CODE) To Kill Your Next Coding Interview

You may hate code challenges and coding interviews but reality is a lot of companies from Google to Amazon do care that you understand the difference between O(n log n) and O(n²), that you do understand when different data structures are approp...

Arrays 21 Binary Tree 26 Bit Manipulation 12

15 Hashing Interview Questions (EXPLAINED) To Check Before Next Coding Interview

Hashing is the practice of using an algorithm (or hash function) to map data of any size to a fixed length. This is called a hash value (or sometimes hash code or hash sums or even a hash digest if you’re feeling fancy). In hashing, keys are converte...

Hash Tables 31

66 MERN Stack Interview Questions (ANSWERED) To Nail Your Next Tech Interview

MERN stands for MongoDB, Express, React, Node, after the four key technologies that make up the stack. MERN is one of several variations of the MEAN stack (MongoDB Express Angular Node), where the traditional Angular.js frontend framework is replaced...

MongoDB 83 Node.js 95 React 164

17 Common Agile Interview Questions For Experienced Software Developers (CLARIFIED)

Agile projects are 28% more successful than traditional projects. Almost 86% of 101,5 international surveyed software developers use Agile in their work. 71% of surveyed companies admitted using Agile approaches sometimes, often, or always. Follow al...

Agile & Scrum 51

10 Unit Testing Interview Questions (CLARIFIED) You'll Be Asked On Next Tech Interview

Unit Tests help you really understand the design of the code you are working on. Instead of writing code to do something, you are starting by outlining all the conditions you are subjecting the code to and what outputs you'd expect from that. Follow ...

Software Testing 26

Guest Voice: The Most Asked Go Interview Questions By ScienceSoft

It has been a long time since the developer community got excited over any new program language. Golang is simply a great initiative made by google and will benefit businesses all over the spectrum. Follow along and check The Most Asked Golang Interv...

Golang 49

18 Cryptography Interview Questions You'll Be Asked On Your Next Tech Interview

The first known evidence of the use of cryptography (in some form) was found in an inscription carved around 1900 BC, in the main chamber of the tomb of the nobleman Khnumhotep II, in Egypt and without asymmetric encryption, the Internet as we know i...

Cryptography 27

Load More...

FullStack.Cafe is a biggest hand-picked collection of top Full-Stack, Coding, Data Structures & System Design Interview Questions to land 6-figure job offer in no time.
