Vigo Webs

Follow 424 Followers About

Frequently asked: ES6 Interview Questions and Answers

Vigo Webs Sep 22, 2018 · 7 min read

Q1. What is ES or ECMAScript or ES6?

When JavaScript was developed, the name itself was chosen for marketing reasons since Java was getting popular around the time. But to standardize the language and its specification, it was submitted to ECMA international (European Computer Manufacturers Association), a body for standardization of information and

communication technology. Eventually the language standardized in ECMA is called ECMAScript of ES. The first edition was released on June 1997.

The ES6 is the sixth edition of the language and was released on June 2015. It was initally known as ECMAScript 6 (ES6) and later renamed to ECMAScript 2015. This edition includes many new features like class, modules, iterators, for/of loop, arrow functions, typed arrays, promises, reflection.

In the next year on June 2016, ECMAScript 6 (ES6) was released and later renamed to ECMAScript 2015. This edition intended to continue the themes of language reform, includes two new features: exponentiation operator (**) and Array.prototype.includes.

And then on June 2017, ECMAScript 2017 (ES2017), the eighth edition released which includes features for concurrency and atomics, syntactic integration with promises (async/await).

Q2. What are some of the features of ES6?

Some of the new features of ES6 are:

- Support for constants (also known as “immutable variables”)
- Block-Scope support for both variables, constants, functions
- Arrow functions
- Extended Parameter Handling
- Template Literals and Extended Literals
- Enhanced Regular Expression
- Destructuring Assignment
- Modules, Classes, Iterators, Generators
- Enhanced Object Properties
- Support for Map/Set & WeakMap/WeakSet
- Promises, Meta-Programming ,Internationalization and Localization

Q3. What is `let` and `const` ? And how it differs from `var` ?

Previously, when we declare any variable using `var`, it was function scoped. Meaning the variable can be accessed within the function. This leads to wrap the code in a function whenever we need to create a new scope.

But `let` and `const` uses block scoping. This means the variable declared using these keywords only exist within the innermost block that surronds them.

```
let a = 5;
{
  let a = 3;
  console.log('inner a:', a); // inner a: 3
}
console.log('outer a: ', a) // outer a: 5;
```

If we declare a `let` variable inside a block like if condition, for loop, it can be accessed within the block.

```
if(true) {
  let i = 0;
  console.log(i); // prints 0;
}
console.log(i); // throws ReferenceError: i is not defined
```

Also we can not re-declare the same variable with the same scope.

```
{
  let a = 0;
  console.log(a); // 0
  let a = 1; //TypeError: Identifier 'a' has already been declared
}
```

Unlike `const`, `const` is immutable. It means the value must be given at the time of the declaration and it can not be re-assigned or changes. Although we can not change the value of the `const` but we can mutate them.

```
{
  const a = 0;
  a = 1; // TypeError: Assignment to constant variable
  const b = [ 1, 2 ];
  b.push(3); // [1, 2, 3]
  b[3] = 4; // [1, 2, 3, 4]
```

```
}
```

It is always a good practice to use `let` and `const` over `var` and if we are not changing the value of the variable we should use `const` .

Q4. What is Arrow function? What are all its uses? How it differs from normal function?

Arrow functions are a short-hand notation for writing functions in ES6. The arrow function definition consists of a parameter list `( ... )` , followed by the `=>` marker and a function body. For single-argument functions, the parentheses may be omitted.

```
// Classical Function Expression

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

// Implementation with arrow function

const add = (a, b) => a + b;

// With single argument, no parentheses required

const add5 = a => 5 + a;
```

If the arrow function is implemented with “concise body” (without `{}`), it does not need an explicit `return` statement. Note the omitted `{ }` after the `=>` .

Arrow functions behavior with `this` keyword varies from that of normal functions. Each function in JavaScript defines its own this context but arrow functions capture the `this` value of the nearest enclosing context.

```
function Timer() {
    this.seconds = 0;
    setInterval(() => {
        setTimeout(() => {
            this.seconds++; // `this` properly refers to the Timer object
        }, 1000);
    }, 1000);
}
```

There are four fundamental differences between arrow functions and `function` functions

- They close over `this`, and do not have their own versions.
- They can have a concise body (without `{ }`) rather than a verbose one (but they can have a verbose body as well).
- They cannot be used as constructors. E.g., you can't use `new` with an arrow function. Hence arrow functions do not have a `protoype` property on them.
- There is no generator syntax for arrow functions. E.g., there is no arrow equivalent to `function *foo() { ... }`.

Q5. What are all the new changes in Object literal notations in ES6?

ES6 allows declaring object literals by providing shorthand syntax for initializing properties from variables and defining function methods. It also enables the ability to have computed property keys in an object literal definition.

```
var a = 'foo', b = 42, c = {};

// shorthand for var o = { a: a, b: b, c: c }
var o = {a, b, c};

// shorthand method
var o = {
   add(a, b) { return a + b }
};

// computed property names
var prop = 'foo';
var o = {
   [prop]: 'bar'
};
// equals var o = { 'foo': 'bar' }
```

Q6. What is `WeakMap` in ES6?

The `WeakMap` is same as `Map` where it is a collection of key/value pairs. But in `WeakMap`, the **keys must be objects** and the values can be arbitrary values. The object references in the keys are held weakly, meaning that they are a target of garbage collection (GC) if

there is no other reference to the object anymore. The `WeakMap` API is the same as the `Map` API.

However, One difference to Map objects is that WeakMap keys are not enumerable. And there are no methods giving us a list of keys. If they were, the list would depend on the state of garbage collection, introducing non-determinism. If we want to have a list of keys, we should use a `Map` .

```
var wm1 = new WeakMap();
var o1 = {};

wm1.set(o1, 37);

wm1.get(o1); // 37

wm1.has(o1); // true

wm3.set(o1, 40);
wm3.get(o1); // 40

wm1.has(o1); // true
wm1.delete(o1);
wm1.has(o1); // false
```

Q7. What is `Set` ?

`Set` objects are collections of unique values. Duplicate values are ignored, as the collection must have all unique values. The values can be primitive types or object references.

```
var mySet = new Set();

mySet.add(1); // Set [ 1 ]
mySet.add(5); // Set [ 1, 5 ]
mySet.add(5); // Set [ 1, 5 ] -- ignored
mySet.add('some text'); // Set [ 1, 5, 'some text' ]
var o = {a: 1, b: 2};
mySet.add(o);

mySet.size; // 4

console.log(mySet); // Set [ 1, 5, 'some text', Object {a: 1, b: 2} ]
```

Also, `NaN` and `undefined` can also be stored in a `Set`. `NaN` is considered the same as `NaN` (even though `NaN !== NaN`).

Q8. What is `Class` expression?

The `Class` expression is one way to define a class in ES6. Similar to function expressions, class expressions can be named or unnamed. If named, the name of the class is local to the class body only. JavaScript classes use prototype-based inheritance.

```
var Rectangle = class {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  area() {
    return this.height * this.width;
  }
}

console.log(new Rectangle(5,8).area());
// expected output: 40
```

A class expression has a similar syntax to a class statement (declaration). However, with class expressions, you are able to omit the class name ("binding identifier"), which you can't with class statements. Additionally, class expressions allow you to redefine/re-declare classes and don't throw any type errors like class declaration. The constructor property is optional. And, typeof the classes generated using this keyword will always be "function".

Q9. What is Generator function?

Generator functions are a new feature in ES6 that allow a function to generate many values over time by returning an object which can be iterated over to pull values from the function one value at a time.

A generator function returns an **iterable object** when it's called. It is written using the new `*` syntax as well as the new yield keyword introduced in ES6.

```
function *infiniteNumbers() {
    let n = 1;
    while (true) {
```

```
        yield n++;
    }
}

const numbers = infiniteNumbers(); // returns an iterable object

numbers.next(); // { value: 1, done: false }
numbers.next(); // { value: 2, done: false }
numbers.next(); // { value: 3, done: false }
```

Each time yield is called, the yielded value becomes the next value in the sequence. Also, note that generators compute their yielded values on demand, which allows them to efficiently represent sequences that are expensive to compute, or even infinite sequences.

Q10. What is `WeakSet` ?

The `WeakSet` object lets you store weakly held objects in a collection.

`WeakSet` objects are collections of objects. An object in the `WeakSet` may occur only once; it is unique in the `WeakSet` 's collection. The main differences to the `Set` object are:

- Unlike `Set` , `WeakSet` s are collections of **objects only** and not of arbitrary values of any type.
- The `WeakSet` is weak: References to objects in the collection are held weakly. If there is no other reference to an object stored in the `WeakSet` , they can be garbage collected. That also means that there is no list of current objects stored in the collection. WeakSets are not enumerable.

```
var ws = new WeakSet();
var window = {};
var foo = {};

ws.add(window);
ws.add(obj);

ws.has(window); // true
ws.has(foo);    // false, foo has not been added to the set

ws.delete(window); // removes window from the set
ws.has(window);    // false, window has been removed
```

For more ES6 Interview Questions and answer use our Android App:

https://play.google.com/store/apps/details?id=com.vigowebs.interviewquestions

JavaScript ES6 Typescript