JavaScript Interview Questions & Answers

Note: Keep in mind that many of these questions are open-ended and could lead to interesting discussions that tell you more about the person's capabilities than a straight answer would.

1. Explain how `this` works in JavaScript ⭐

▼ Answer

A function's `this` keyword behaves a little differently in JavaScript compared to other languages. It also has some differences between strict mode and non-strict mode.

In the **global execution context (outside of any function)**, `this` refers to the global object whether in `strict mode` or not.

Inside a function, the value of this depends on how the function is called.

Implicitly Binding: As an object method its `this` is set to the object the method is called on.

Explicit Binding: Functions have three methods on their prototype, bind, call, and apply. If a function is called with these methods, then `this` is set to the first argument passed.

As an example:

```
function echoThis() {
  console.log(this);
}
echoThis.call('hello'); // hello
```

new Binding: If a function is called using the `new` keyword, an empty object is created and assigned to `this` inside the function.

default Binding: If a function is called, but the three scenarios above do not apply, then `this` is set to the global object if not in strict mode, and `undefined` if in strict mode.

Arrow function exception: If a function is defined as an arrow function, the prior rules will not apply. Instead, `this` will refer to the `this` binding in the immediate scope where the arrow function was declared.

References

- MDN web docs / this

2. What is the difference between `let` and `var` and `const` ? ⭐

▼ Answer

`const` is a signal that the identifier won't be reassigned. It needs initialization upfront, so you can't write const something;

`let` is a signal that the variable may be reassigned, such as a counter in a loop, or a value swap in an algorithm.

`var` is now the weakest signal available when you define a variable in JavaScript. The variable may or may not be reassigned, and the variable may or may not be used for an entire function, or just for the purpose of a block or loop. It's declaration is hoisted, instead of `let` and `const`.

```
for (var i = 0; i < 2; i++) {}
console.log(i); // exists outside the block scope
for (let i = 0; i < 2; i++) {}
console.log(i); // only exists inside the block scope
for (const i = 0; i < 2; i++) {}
console.log(i); // error reassignment, but only on top-level
for (const cnt = { i: 0 }; cnt.i < 2; cnt.i++) {} // only exists inside the block scope
```

3. What is `===` operator? ⭐

▼ Answer

This is the strict comparison operator e.g. `5 == '5' = true` VS `5 === '5' = false`, this means that it checks the value and also the type, so that Int 5 isn't equal a Str 5.

4. Explain the difference between "==" and "==="? ⭐

▼ Answer

"==" checks only for equality in value whereas "===" is a stricter equality test and returns false if either the value or the type of the two variables are different.

5. List out the different ways an HTML element can be accessed in a Javascript code. ⭐

▼ Answer

Access one element:

```
let byID = document.getElementById('id');
let qS = document.querySelector('#id');
```

They return the first matching node. querySelector is the new selector interface, should be faster, but depends on browser implementation. querySelector can take any css-selector and is more comfortable.

Access one and more:

```
let byClass = document.getElementsByClassName(classname);
let qSA = document.querySelectorAll('.classname');
```

They return a non-live NodeList, which is an array-like list of elements, array-like means that some functions are missing like push(), pop()).

6. What does a `typeof` operator do? ⭐

▼ Answer

The `typeof` operator is used to get the data type (returns a string) of its operand. The operand can be either a literal or a data structure such as a variable, a function, or an object. The operator returns the data type.

Syntax:

```
typeof operand;
typeof operand;
```

7. What is the difference between Local Storage and Session Storage? ⭐

▼ Answer

LocalStorage

- It can store up to 10Mb offline data.
- The data is not sent back to the server for every HTTP request (HTML, images, JavaScript, CSS, etc) - reducing the amount of traffic between client and server.
- The data stored in localStorage persists until explicitly deleted. Changes made are saved and available for all current and future visits to the site.
- It works on same-origin policy. So, data stored will only be available on the same origin.

SessionStorage

- It is similar to localStorage.
- The data is not persistent i.e. data is only available per window (or tab in browsers like Chrome and Firefox). Data is only available during the page session. Changes made are saved and available for the current page, as well as future visits to the site on the same window. Once the window is closed, the storage is deleted.
- The data is available only inside the window/tab in which it was set.
- Like localStorage, tt works on same-origin policy. So, data stored will only be available on the same origin.

For more info please check MDN - LocalStorage & MDN - SessionStorage

8. What is the difference between `null` and `undefined` ? ⭐

▼ Answer

`null` and `undefined` are two types in JavaScript. `undefined` means something hasn't been initialized. `null` means something is currently unavailable.

9. What are anonymous functions in Javascript? ⭐

▼ Answer

Anonymous functions (also called lambda functions) are functions where the name is omitted. They are commonly used as parameters to other functions or stored in a variable.

```
//common use
setTimeout(function() {
    console.log('Hi from my anonymous function');
}, 300);

// double arrow function
setTimeout(() => {
    console.log('Hi from my anonymous function');
}, 300);

// assigning to a variable
const myFunc = () => {
    // do something awesome
}
```

References

- helephant.com / js-anonymous-function

10. What is a function callback? ⭐

▼ Answer

A callback function is a function that is passed to another function as an argument and is executed after some operation has been completed. Below is an example of a simple callback function that logs to the console after some operations have been completed.

```
const modifyArray = (arr, callback) => {
  // do something to arr here
  arr.push(100);

  // then execute the callback function that was passed
  callback();
};

var arr = [1, 2, 3, 4, 5];

modifyArray(arr, function() {
  console.log('array has been modified', arr);
});
```

11. What is the difference between `innerHTML` and `innerText` ? ⭐

▼ Answer

`innerHTML` lets you work with HTML rich text and doesn't automatically encode and decode text. In other words, `innerText` retrieves and sets the content of the tag as plain text, whereas `innerHTML` retrieves and sets the content in HTML format.

12. What is the difference between `HTMLCollection` and `NodeList` ? ⭐

▼ Answer

Both `HTMLCollection` and `NodeList` are collections of DOM nodes. Specifically, an `HTMLCollection` is a collection of Elements which can be accessed by either index or the element's name or id attributes. A `NodeList` is a collection of nodes and is an interface representing an ordered collection without specifying a particular implementation. Elements in an `HTMLCollection` are always live (they are updated in the collection when the underlying document is updated) while items in a `NodeList` may be live or static depending on which method is used to retrieve it.

`HTMLCollection` s and `NodeList` s also differ in the methods they provide; an `HTMLCollection` provides a `namedItem` method to access elements by `name` or `id` attribute while a `NodeList` provides methods to access the collection's `keys` and `values` .

References

- MDN - HTMLCollection
- MDN - NodeList
- HackerNoon - HTMLCollection, NodeList and array of objects

13. What is an Event Bubbling in Javascript? ⭐

▼ Answer

When an event happens on an element, it first runs the handlers on it, then on its parent, then all the way up on other ancestors.
Event bubbling is a type of event propagation where the event first triggers on the innermost target element, and then successively triggers on the ancestors of the target element in the same nesting hierarchy till it reaches the outermost DOM element or document object.

Let's say, we have 3 nested elements `FORM` > `DIV` > `P` with a handler on each of them:

```
<form onclick="alert('form')">FORM
<div onclick="alert('div')">DIV
    <p onclick="alert('p')">P</p>
</div>
</form>
```

A click on the inner `<p>` first runs onclick:

1. On that `<p>`.
2. Then on the outer `<div>`.
3. Then on the outer `<form>`.
4. And so on upwards till the document object.

> So if we click on `<p>`, then we'll see 3 alerts. The process is called "bubbling", because of events "bubble" from the inner element up through parents like a bubble in the water.

For more info & reference Javascript - Bubbling and capturing

14. What is NaN in Javascript? ⭐

▼ Answer

The global `NaN` property is a value representing Not-A-Number.

15. Which Test-Libraries do you use? ⭐

▼ Answer

q-unit, mocha, chai, sinonJS, jasmine, ...

16. What is Hoisting? ⭐

▼ Answer

Means that the declaration moved to the top of the current scope (current script or the current function). JavaScript only hoists declarations, not initializations.

`let` and `const` don't get hoisted.

17. What means `use strict` ? ⭐

▼ Answer

Switches to strict mode which helps to prevent common errors like using unsafe operators

18. What is `bind()` and when we use it? ⭐⭐

▼ Answer

`bind` is a method to bind the current context for later execution e.g.

```
element.addEventListener('click', this.onClick.bind(this), false);
```

it creates a new function which prevents accidental loss of scope. An alternative approach is to use apply, call or ES6 fat-arrow function.

19. Explain higher order function? ⭐⭐

▼ Answer

Function that will take a function as argument or return a new function. For example `[].map/filter/reduce` are higher order functions.

20. Explain `map`, `filter`, `reduce` and when to use it? ⭐

▼ Answer

`map` - to iterate over an array and return a new one

`filter` - to filter an array and return a new filtered one

`reduce` - takes and reducer function which evaluate against every element and can produce every desired output (filter, map or simple value like sum)

21. When to use event delegation? ⭐⭐

▼ Answer

If you have to watch a lot of elements and performance is key

22. What is the difference between ViewState and SessionState? ⭐

▼ Answer

`ViewState` is specific to a page in a session.

`SessionState` is specific to user specific data that can be accessed across all pages in the web application.

23. Does JavaScript support automatic type conversion?

▼ Answer

Yes! JavaScript does support automatic type conversion.

24. What are IIFE in javascript?

▼ Answer

An IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined. It is a design pattern which is also known as a Self-Executing Anonymous Function.

```
//example
(function () {
    var name = "John Doe";
})();
// Variable name is not accessible from the outside scope
name // throws "Uncaught ReferenceError: name is not defined"

//common use
var result = (function () {
    var name = "John Doe";
    return name;
})();
// Immediately creates the output:
result; // "John Doe"
```

Assigning the IIFE to a variable stores the function's result, not the function itself.

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.