20 Important JavaScript Concepts for Your Next Interview

1. Closures

A closure is a function that remembers its outer variables even after the outer function has finished executing.

```
function outer() {
  let count = 0;
  return function inner() {
    count++;
    return count;
  };
}

const counter = outer();
console.log(counter()); // 1
console.log(counter()); // 2
```

2. Hoisting

In JavaScript, variable and function declarations are "hoisted" to the top of their scope.

```
console.log(greet()); // Hello!

function greet() {
  return "Hello!";
}

console.log(num); // undefined
var num = 5;
```

3. Event Loop & Callbacks

JavaScript is single-threaded, and the event loop allows asynchronous operations using callbacks.

```
console.log("Start");
setTimeout(() => console.log("Async operation"), 1000);
console.log("End");

// Output: Start, End, Async operation
```

4. Promises

Promises handle async operations, with states: pending, fulfilled, and rejected.

```
async function fetchData() {
  let data = await new Promise(resolve => setTimeout(() => resolve("Data"), 1000));
  console.log(data);
}

fetchData(); // Data
```

5. Async/Await

Async/await simplifies promise handling.

```
async function fetchData() {
  let data = await new Promise(resolve => setTimeout(() => resolve("Data"), 1000));
  console.log(data);
}

fetchData(); // Data
```

6. Arrow Functions

Arrow functions provide a concise syntax and don't have their own this.

```
const add = (a, b) => a + b;
console.log(add(2, 3)); // 5
```

7. Destructuring

Destructuring allows you to unpack values from arrays or properties from objects.

```
const person = { name: "Alice", age: 25 };
const { name, age } = person;

console.log(name); // Alice
console.log(age); // 25
```

8. Spread & Rest Operators

Spread ... expands elements, and
Rest collects them into an array.

```
const arr1 = [1, 2, 3];
const arr2 = [...arr1, 4, 5]; // Spread

function sum(...nums) { // Rest
  return nums.reduce((a, b) => a + b);
}
console.log(sum(1, 2, 3, 4)); // 10
```

9. Prototypes

Prototypes allow objects to inherit
properties and methods.

```
function Car(name) {
  this.name = name;
}

Car.prototype.getName = function() {
  return this.name;
};

const myCar = new Car("Tesla");
console.log(myCar.getName()); // Tesla
```

10. This Keyword

this refers to the context in which a function is called.

```
const person = {
  name: "John",
  sayName() {
    console.log(this.name);
  },
};

person.sayName(); // John
```

11. Classes

ES6 classes provide a cleaner syntax for object-oriented programming.

```
class Animal {
  constructor(name) {
    this.name = name;
  }
  speak() {
    return `${this.name} makes a sound.`;
  }
}

const dog = new Animal("Dog");
console.log(dog.speak()); // Dog makes a sound.
```

12. Modules

Modules let you split your code across multiple files.

```js
// add.js
export const add = (a, b) => a + b;

// main.js
import { add } from "./add.js";
console.log(add(2, 3)); // 5
```

13. Map and Filter

map and filter are array methods for transforming and filtering arrays.

```js
const numbers = [1, 2, 3, 4];
const doubled = numbers.map(n => n * 2); // [2, 4, 6, 8]
const evens = numbers.filter(n => n % 2 === 0); // [2, 4]
```

14. Reduce

reduce accumulates values from an array.

```
const numbers = [1, 2, 3, 4];
const sum = numbers.reduce((acc, n) => acc + n, 0);
console.log(sum); // 10
```

15. SetTimeout & SetInterval

setTimeout delays execution, while setInterval repeats it.

```
setTimeout(() => console.log("After 1 second"), 1000);

let count = 0;
const intervalId = setInterval(() => {
  console.log("Count:", ++count);
  if (count === 3) clearInterval(intervalId);
}, 1000);
```

16. Template Literals

Template literals allow multi-line strings and interpolation.

```
const name = "World";
console.log(`Hello, ${name}!`); // Hello, World!
```

17. Type Coercion

JavaScript can implicitly convert types, sometimes unpredictably.

```
if ("") {
  console.log("This won't run");
} else {
  console.log("Falsy value");
}
```

18. Truthy & Falsy Values

Values like 0, "", null, undefined, NaN are falsy.

```
if ("") {
  console.log("This won't run");
} else {
  console.log("Falsy value");
}
```

19. Debouncing & Throttling

Debouncing and throttling are techniques to control function execution frequency, often in response to events.

Debounce (delay execution):

```
function debounce(func, delay) {
  let timeout;
  return function (...args) {
    clearTimeout(timeout);
    timeout = setTimeout(() => func.apply(this, args), delay);
  };
}

window.addEventListener("resize", debounce(() => console.log("Resized!"), 500));
```

Throttle (limit execution):

```
function throttle(func, limit) {
  let inThrottle;
  return function (...args) {
    if (!inThrottle) {
      func.apply(this, args);
      inThrottle = true;
      setTimeout(() => (inThrottle = false), limit);
    }
  };
}

window.addEventListener("scroll", throttle(() => console.log("Scrolling!"), 200));
```

20. Currying

Currying transforms a function with multiple arguments into a series of functions with a single argument.

```
function multiply(a) {
  return function (b) {
    return a * b;
  };
}

const double = multiply(2);
console.log(double(5)); // 10
```

Found this helpful?

Follow for more!

Sanuj Bansal