#_ Common JavaScript Interview Questions

1. What is JavaScript?

Answer: JavaScript is a high-level, interpreted programming language used to enhance web pages. It adds interactivity to the site and is an essential part of web development.

---

2. Explain the Difference Between == and ===?

Answer: The == operator compares values for equality, converting them to a common type. The === operator compares both value and type, without performing any type conversion.

```
'5' == 5   // true
'5' === 5  // false
```

---

3. Datatypes of JavaScript?

Answer: JavaScript supports several data types, which can be broadly categorized into two types: Primitive and Non-Primitive (or Reference).

Primitive data types include:

- String
- Number
- Boolean
- Undefined
- Null
- BigInt (as of ES2020)
- Symbol (introduced in ES6)

Non-Primitive (Reference) data types:

- Object
- Array

- Function

---

4. Difference between primitive vs non-primitive?

Answer: Primitive types are the simplest form of data types and are immutable (cannot be changed). They represent a single value and are stored directly on the stack.

Non-primitive types (Reference types) are more complex types like arrays, objects, and functions. They are stored in the heap, with a reference to their location being stored on the stack. Because of this reference mechanism, they are mutable.

---

5. What are arrays, functions, and objects?

Answer:

- **Arrays:** Arrays are a type of object used for storing multiple values in a single variable. Each value (element) in an array has a numeric position, known as its index.
- **Functions:** Functions are reusable blocks of code that perform a specific task. They can take inputs (parameters) and return an output (value).
- **Objects:** Objects are collections of key-value pairs. They can store data and methods to act upon that data.

---

5. What is scope in JavaScript?

Answer: Scope in JavaScript refers to the context in which a variable exists. It determines the accessibility and visibility of variables, functions, and objects. There are mainly two types of scopes:

- **Local Scope (Function Scope):** Variables defined within a function are only accessible within that function.

- **Global Scope:** Variables defined outside any function are accessible from any other code in the entire program.

---

6. Difference between Var, Let, and const in JavaScript?

Answer:

- **Var:** It's function-scoped and can be re-declared and updated.
- **Let:** Introduced in ES6, it's block-scoped and can be updated but not re-declared.
- **Const:** Also introduced in ES6, it's block-scoped but cannot be re-declared or updated.

---

7. What is a Loop?

Answer: A loop is a programming concept used to execute a set of statements repeatedly based on a condition.

---

8. Difference between for, while, and do-while loops?

Answer:

- **For loop:** Iterates a set number of times. It has three parts: initialization, condition, and increment/decrement.
- **While loop:** Iterates as long as a specified condition is true. The condition is evaluated before each iteration.
- **Do-while loop:** Like the while loop but checks the condition after the loop has executed, ensuring the loop is executed at least once.

---

9. Difference between for..of and for..in loop?

Answer:

- **For..in:** Iterates over enumerable properties of an object, typically used for objects.
- **For..of:** Iterates over iterable objects (like arrays, strings, etc.), introduced in ES6.

---

10. What is the forEach method?

Answer: `forEach` is a method on the Array prototype that iterates over each element in the array. It takes a callback function that gets executed for each element.

---

11. Types of functions in JavaScript?

Answer:

- **Named Functions:** Have a name and can be called using that name.
- **Anonymous Functions:** Have no name and are typically used as arguments to other functions.
- **Immediately Invoked Function Expressions (IIFE):** Runs as soon as it's defined.
- **Arrow Functions:** Introduced in ES6, they have a concise syntax and do not bind their own `this`.

---

12. What is function expression in JavaScript?

Answer: A function expression is when a function is assigned to a variable. It can be named or anonymous. The function can be invoked using the variable name.

13. What are callback functions?

Answer: A callback function is a function that is passed as an argument to another function and is executed after the outer function has completed.

---

14. When to use callback function in real application?

Answer: Callbacks are used for asynchronous operations like reading files, making API calls, or any operation that doesn't yield the result immediately. They help in executing code after a certain operation completes without blocking the main thread.

---

15. What is the use of Event Handling in JavaScript?

Answer: Event handling in JavaScript is used to manage and handle events, like clicks or key presses, to provide dynamic interactivity in web applications.

---

16. What is a Higher-Order Function?

Answer: Higher-order functions are functions that take one or more functions as arguments, return a function, or both. Examples include `map`, `filter`, and `reduce`.

---

17. Explain Closures in JavaScript.

Answer: A closure is a function having access to its own scope, the scope of the outer function, and the global scope. They are used to create private variables or functions.

```
function outer() {
  let count = 0;
  return function inner() {
    count++;
    return count;
  };
}
const counter = outer();
console.log(counter()); // 1
```

---

18. Explain Hoisting in JavaScript.

Answer: Hoisting is JavaScript's behavior where variables and function declarations are moved or "hoisted" to the top of their containing scope during compilation. For variables declared with `var`, they are hoisted and initialized with `undefined`. In contrast, variables declared with `let` and `const` are hoisted but remain uninitialized.

---

19. What is a Promise?

Answer: A Promise represents a future value. It has three states - pending, fulfilled, and rejected. Promises allow handling asynchronous operations more gracefully.

```
let promise = new Promise((resolve, reject) => {
  // Asynchronous code here
});
```

---

20. Explain Event Delegation.

Answer: Event delegation allows you to add an event listener to a parent element instead of adding it to the child elements individually. It utilizes the concept of event bubbling in the DOM.

---

21. What is the 'this' keyword?

Answer: In JavaScript, `this` refers to the object it belongs to. Its value can change depending on the context in which it's called.

---

22. How does the `this` keyword work in arrow functions?

Answer: Arrow functions don't have their own `this`. Instead, they inherit `this` from the enclosing function or context. This behavior makes arrow functions particularly suitable for callbacks and event handlers, especially within frameworks and libraries where preserving the context of `this` is important.

---

23. What is the event loop and the call stack in JavaScript?

Answer: The event loop is the continuous process that the JavaScript engine uses to monitor the call stack and the callback queue. If the call stack is empty, it takes the first event from the queue and pushes it to the call stack to be executed. The call stack is a data structure that tracks the execution of functions in a program. Functions get added (pushed) to the stack as they are called and removed (popped) as they complete.

---

24. What is a `Set` in JavaScript? How is it different from an array?

Answer: A `Set` is a built-in object in JavaScript that allows you to store unique values of any type. The main differences between a `Set` and an Array are:

- Every value in a `Set` must be unique; an array can have duplicate values.

- Set does not have methods like pop or push but has add, delete, and has methods.
- Set does not have indexed access to elements like an array.

---

25. Explain the Prototype Chain.

Answer: The prototype chain is the mechanism by which objects in JavaScript inherit properties from one another. If a property is not found on an object, JavaScript will look up the prototype chain to find it.

---

26. How to Deep Clone an Object?

Answer: Deep cloning means creating a new object that is a copy of an existing object with all nested properties copied. A common way to achieve this is using JSON:

```
let newObj = JSON.parse(JSON.stringify(oldObj));
```

---

27. Explain Callback Functions.

Answer: A callback function is a function passed into another function as an argument, which is then invoked inside the outer function.

---

28. How do you create a Class in JavaScript?

Answer: You can create a class using the class keyword. Classes are used to create objects and encapsulate behavior.

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

---

29. Explain the difference between null and undefined.

Answer: null is an assignment value that represents no value or no object, whereas undefined means that a variable has not been initialized.

---

30. What are Arrow Functions?

Answer: Arrow functions are a new ES6 syntax for writing JavaScript functions. They are more concise and do not have their own bindings to this, arguments, super, or new.target.

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

---

31. What are Template Literals?

Answer: Template literals allow embedded expressions and create string literals allowing embedded expressions.

```
let name = 'John';
console.log(`Hello, ${name}!`); // Hello, John!
```

---

32. What are Asynchronous operations in JavaScript?

Answer: Asynchronous operations allow tasks to be executed outside the main program flow, enabling the program to continue running in the meantime. Examples include AJAX calls, reading files, and timers.

---

33. How to implement Promises in JS?

Answer: A promise is created using the Promise constructor. It takes a single callback function as an argument, which, in turn, takes two functions as parameters: resolve and reject.

```
let myPromise = new Promise((resolve, reject) => {
    // some logic
    if (/* everything worked */) {
        resolve('Success!');
    } else {
        reject('Failure!');
    }
});
```

---

34. When to use Promises in real applications?

Answer: Promises are used to handle asynchronous operations more gracefully, providing a more concise and organized way to work with async code. They are used when you want to perform some task that takes time and you want to execute something once it's completed, without blocking the main thread.

---

35. Explain Async/Await.

Answer: Async/Await is a way to handle asynchronous code. An async function returns a promise, and await waits for the promise to resolve or reject.

```
async function fetchData() {
  let response = await fetch('url');
  let data = await response.json();
  return data;
}
```

---

36. What is destructuring?

Answer: Destructuring allows unpacking values from arrays or properties from objects into distinct variables.

```
const person = {name: 'John', age: 30};
const {name, age} = person;
```

---

37. How do you implement Inheritance in JavaScript?

Answer: Inheritance can be implemented using the extends keyword to create a subclass.

```
class Animal {
  // ...
}
class Dog extends Animal {
  // ...
}
```

---

38. What is Strict Mode in JavaScript?

Answer: Strict mode makes several changes to normal JavaScript semantics. For example, it eliminates some silent errors by changing them to throw errors.

---

39. How to check if a property exists in an object?

Answer: You can use the in operator or the hasOwnProperty method.

```
if ('property' in object) {
  // ...
}
if (object.hasOwnProperty('property')) {
  // ...
}
```

40. How do you create a private variable in JavaScript?

Answer: You can use closures or Symbols to create private variables.

---

41. What are the different ways to create an object in JavaScript?

Answer: You can create objects using object literals, constructors, Object.create method, and classes.

---

42. Explain JavaScript timers.

Answer: Timers like setTimeout and setInterval are used to execute code after a set time interval.

```
setTimeout(() => {
  console.log('Runs after 2 seconds');
}, 2000);
```

---

43. Explain the spread operator.

Answer: The spread operator allows an iterable to be expanded where zero or more arguments are expected.

```
let arr = [1, 2, 3];
let arr2 = [...arr, 4, 5]; // [1, 2, 3, 4, 5]
```

---

44. How do you handle errors in JavaScript?

Answer: Errors can be handled using try-catch-finally blocks.

```
try {
  // code that may throw an error
} catch (error) {
  // handling the error
} finally {
  // code to run regardless of an error
}
```

---

45. What are Symbols in JavaScript?

Answer: Symbols provide a way to create unique and immutable values, often used for object property keys.

---

46. Explain JavaScript Modules.

Answer: JavaScript modules allow separating code into multiple files and managing dependencies through export and import statements.

---

47. What are JavaScript Generators?

Answer: Generators are special functions that can pause and resume their execution, allowing other code to run in between.

```
function* generator() {
  yield 'value';
}
```

---

48. How do you check if a number is an integer?

Answer: You can use `Number.isInteger(value)` to check if a value is an integer.

---

49. What is the difference between a method and a function?

Answer: A method is a function associated with an object, whereas a function is a standalone unit of code.

---

50. What are Promises in a JavaScript?

Answer: Promises provide a way to handle asynchronous operations, representing a value that may not be available yet but will be resolved or rejected in the future.

---

51. What are JavaScript Service Workers?

Answer: Service workers are scripts that run in the background, separate from the web page, allowing features like offline access, push notifications, and background sync.

---

52. Explain Tail Call Optimization.

Answer: Tail call optimization reduces the call stack size for recursive functions by reusing the current stack frame if a function's return value is the same as the last call.

---

53. How to create a memory leak in JavaScript?

Answer: Memory leaks can be created by unintentionally retaining references to objects, like circular references between objects.

---

54. What is the importance of Webpack in modern web development?

Answer: Webpack is a module bundler and build tool that helps manage dependencies, transpile code, and optimize assets for better performance.

---

55. How does CORS work?

Answer: CORS (Cross-Origin Resource Sharing) is a security feature implemented by browsers to restrict web pages from making requests to a domain different than the one that served the web page.

---

56. What is a JavaScript Proxy?

Answer: A Proxy object wraps another object and intercepts operations, like reading and writing properties.

```
let proxy = new Proxy(target, handler);
```

---

57. How to flatten a deeply nested array?

Answer: You can use recursion or the `flat` method with an appropriate depth argument.

```
let arr = [1, [2, [3, [4]]]];
let flattened = arr.flat(Infinity);
```

---

58. How to cancel a fetch request?

Answer: You can use the `AbortController` to signal a fetch request to be aborted.

```
const controller = new AbortController();
fetch(url, { signal: controller.signal });
controller.abort(); // Cancels the request
```

---

59. Explain the difference between debounce and throttle techniques.

Answer: Debouncing delays invoking a function until after a certain amount of time has passed since the last time the function was invoked. Throttling limits the maximum number of times a function can be called over time.

---

60. What is the purpose of the JavaScript `bind` method?

Answer: The `bind` method creates a new function that, when called, has its `this` value set to the provided value.

---

61. What are Observables in JavaScript?

Answer: Observables are lazy Push collections of multiple values, used to handle asynchronous operations. They are part of the RxJS library.

---

62. What is the difference between the Document Object Model (DOM) and Virtual DOM?

Answer: The DOM is a representation of the web page. The Virtual DOM is a concept used in some modern UI libraries to efficiently update the DOM by minimizing direct manipulations.

---

63. What is a Polyfill?

Answer: A polyfill is code that implements features on web browsers that do not support those features.

---

64. What are JavaScript Mixins?

Answer: A mixin is a class containing methods that can be used by other classes without a need for inheritance.

---

65. How do you prevent code blocking in JavaScript?

Answer: You can prevent code blocking using asynchronous techniques like callbacks, promises, async/await, and Web Workers.

---

66. Explain Immutability.

Answer: Immutability refers to the concept that an object should not be modified after it's created. Instead of modifying, new objects are created with the desired changes.

---

67. Can you explain how prototypal inheritance works in JavaScript?

Answer: In JavaScript, each object can have another object as its prototype. When trying to access a property that does not exist in an object, JavaScript tries to find this property in the object's prototype, and its prototype, and so on, until it reaches the end of the prototype chain. This behavior is the basis for prototypal inheritance where an object can inherit properties and methods from another object.

68. Can you explain the difference between null and undefined?

Answer: Both null and undefined represent the absence of value in JavaScript. However, they are used in slightly different scenarios. undefined typically indicates that a variable has been declared but not assigned a value. On the other hand, null is an intentional assignment indicating no value. Essentially, undefined is the absence of assignment, while null is an explicit assignment of 'no value'.

---

69. What are JavaScript generators?

Answer: Generators are special types of functions in JavaScript that allow the execution flow to be paused and resumed. They are defined using the function* syntax and use the yield keyword to pause/resume. Generators return an iterator object with a next() method, which when called, resumes the execution until the next yield.