207 Preguntas de JS para entrevistas técnicas

Bueno, por si no me conoces soy Víctor Robles y me dedico al desarrollo web desde hace muchos años.

En este libro he recopilado muchas preguntas de JavaScript que, según mi experiencia, se hacen en entrevistas técnicas para puestos de desarrollo web...

Y quizás leerlo antes de tu próxima entrevista puede ayudarte a superarla, quien sabe.

Incluso, si estás estudiando programación en un ciclo formativo o en la universidad te las puedes encontrar algún examen de JS.

Vamos con ellas...

0. ¿Qué es JavaScript y para qué se utiliza?

JavaScript es un lenguaje de programación que se utiliza principalmente para crear contenido interactivo en las páginas web, como animaciones, validación de formularios y manipulación del DOM.

---

1. ¿Quién creó JavaScript y en qué año?

JavaScript fue creado por Brendan Eich en 1995 mientras trabajaba en Netscape Communications.

---

2. ¿Cuál es la principal diferencia entre JavaScript y Java?

Java es un lenguaje de programación orientado a objetos, mientras que JavaScript es un lenguaje de secuencias de comandos basado en eventos y orientado a objetos de forma más flexible.

---

3. ¿Qué significa ECMAScript y cómo se relaciona con JavaScript?

ECMAScript es el estándar en el que se basa JavaScript, definiendo sus características y funcionalidades.

---

4. ¿Qué versión de ECMAScript introdujo las funciones de flecha en JavaScript?

Las funciones de flecha fueron introducidas en ECMAScript 6 (ES6) en 2015.

---

5. ¿Qué es un motor de JavaScript y cuál es su función?

Un motor de JavaScript es un software que ejecuta código JavaScript, interpretando y ejecutando las instrucciones dentro del navegador o en el servidor.

---

6. ¿Qué evento marcó el comienzo de la popularidad de JavaScript en los navegadores web?
El lanzamiento de Netscape Navigator 2.0 con JavaScript integrado en 1995 fue crucial para la adopción de JavaScript.

---

7. ¿En qué se diferencia JavaScript de JScript?
JScript es la implementación de JavaScript de Microsoft, que fue utilizada en versiones anteriores de Internet Explorer, mientras que JavaScript sigue siendo el estándar más ampliamente adoptado.

---

8. ¿Qué significa "JavaScript es un lenguaje interpretado"?
Significa que el código JavaScript es ejecutado línea por línea por un intérprete, sin necesidad de ser compilado previamente.

---

9. ¿Cuál es la principal función de la consola en JavaScript?
La consola permite mostrar mensajes, depurar código y realizar pruebas de manera interactiva durante el desarrollo.

---

10. ¿Qué es el "Document Object Model" (DOM) y cómo se relaciona con JavaScript?
El DOM es una representación estructurada de los documentos HTML y XML que permite a JavaScript interactuar con ellos y modificarlos dinámicamente.

---

11. ¿Qué es el "hoisting" en JavaScript?
El hoisting es un comportamiento de JavaScript en el que las declaraciones de variables y funciones se mueven al inicio de su contexto de ejecución antes de ser ejecutadas.

---

12. ¿Qué fue lo que motivó la creación de JavaScript?
JavaScript fue creado inicialmente para añadir interactividad a las páginas web y mejorar la experiencia del usuario en el navegador.

---

13. ¿Qué es el "strict mode" en JavaScript?
El "strict mode" es un modo de ejecución de JavaScript que permite una evaluación más rigurosa del código, evitando errores comunes y mejorando la seguridad.

---

14. ¿Cuál fue la versión de ECMAScript que introdujo las promesas en JavaScript?
Las promesas fueron introducidas en ECMAScript 6 (ES6).

---

15. ¿Qué es Node.js y cómo se relaciona con JavaScript?
Node.js es un entorno de ejecución de JavaScript del lado del servidor que permite ejecutar código JavaScript fuera de un navegador web.

---

16. ¿Qué significa "asincronía" en JavaScript?
La asincronía en JavaScript permite que ciertas operaciones, como las solicitudes HTTP, se ejecuten en segundo plano sin bloquear el hilo principal del programa.

---

17. ¿Cuál es el propósito de la función "setTimeout" en JavaScript?
La función "setTimeout" se utiliza para ejecutar una función después de un período específico de tiempo.

---

18. ¿Qué es un callback en JavaScript?
Un callback es una función que se pasa como argumento a otra función y se ejecuta después de que esta haya terminado su tarea.

---

19. ¿Qué es JSON y cómo se utiliza en JavaScript?
JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero y fácil de leer, utilizado para enviar y recibir datos entre el servidor y el cliente.

---

20. ¿Qué es la diferencia entre '==' y '===' en JavaScript?
'==' compara los valores de manera flexible (permitiendo conversiones de tipo), mientras que '===' compara tanto los valores como los tipos de datos de manera estricta.

---

21. ¿Cuáles son los tipos de datos primitivos en JavaScript?
Los tipos de datos primitivos en JavaScript son: string, number, boolean, null, undefined, symbol, y bigint.

---

22. ¿Qué es un tipo de dato "primitive" en JavaScript?
Un tipo de dato primitivo en JavaScript es un valor

inmutable que no puede ser alterado una vez creado. Ejemplos son los números, cadenas y booleanos.

---

23. ¿Cuál es la diferencia entre null y undefined en JavaScript?
null representa la ausencia intencionada de un valor, mientras que undefined indica que una variable ha sido declarada pero no inicializada.

---

24. ¿Qué tipo de dato es el valor true en JavaScript?
El valor true es de tipo boolean, que es uno de los tipos de datos primitivos en JavaScript.

---

25. ¿Cómo se puede convertir un valor de tipo string a number en JavaScript?
Se puede convertir un string a number utilizando funciones como parseInt(), parseFloat() o el operador +.

---

26. ¿Qué es un "object" en JavaScript y cómo se diferencia de los tipos primitivos?
Un "object" es una colección de propiedades y métodos, y a diferencia de los tipos primitivos, los objetos son mutables y pueden contener más de un valor.

27. ¿Qué es un "array" en JavaScript y cómo se clasifica como tipo de dato?
Un "array" es un tipo especial de objeto en JavaScript, usado para almacenar colecciones de elementos, y se clasifica como un tipo de dato de objeto.

---

28. ¿Qué significa que los tipos de datos en JavaScript sean "dinámicamente tipados"?
Que una variable puede contener valores de diferentes tipos durante su ciclo de vida, y el tipo de dato se determina en tiempo de ejecución.

---

29. ¿Cuál es la función de typeof en JavaScript?
typeof es un operador utilizado para obtener el tipo de dato de una variable o valor, devolviendo una cadena que describe el tipo.

---

30. ¿Qué es el tipo de dato symbol en JavaScript?
symbol es un tipo de dato primitivo único e inmutable utilizado principalmente para propiedades de objetos que deben ser únicas.

---

31. ¿Cómo se declara una variable de tipo bigint en JavaScript?
Se declara utilizando el sufijo n, como por ejemplo let largeNumber = 123456789012345678901234567890n;.

---

32. ¿Qué tipo de datos es el valor [ ] en JavaScript?
El valor [ ] es un array, que es un tipo de dato objeto en JavaScript.

---

33. ¿Qué tipo de dato es el valor {} en JavaScript?
El valor {} es un objeto vacío, que es un tipo de dato de objeto en JavaScript.

---

34. ¿Cómo se pueden mezclar diferentes tipos de datos en una variable en JavaScript?
Puedes mezclar tipos de datos dentro de un array u objeto, ya que estos pueden contener diferentes tipos de valores.

---

35. ¿Qué tipo de dato devuelve parseInt('123abc') en JavaScript?
Devuelve el número 123 ya que parseInt() convierte el inicio de un string válido a un número.

---

36. ¿Por qué NaN es considerado un valor numérico en JavaScript?

NaN (Not-a-Number) es un valor numérico especial que indica que una operación matemática no ha dado como resultado un valor numérico válido.

---

37. ¿Cómo verificar si un valor es NaN en JavaScript?

Para verificar si un valor es NaN, se utiliza la función Number.isNaN(), ya que el operador == o === no puede comprobar correctamente NaN.

---

38. ¿Qué diferencia hay entre == y === al comparar tipos de datos en JavaScript?

== realiza una comparación de igualdad con conversión de tipos, mientras que === realiza una comparación estricta sin conversión de tipos.

---

39. ¿Cuál es el tipo de dato devuelto por null == undefined?

El tipo de dato devuelto es true, ya que en JavaScript, null y undefined son considerados iguales al compararlos con ==.

40. ¿Por qué los objetos en JavaScript son considerados tipos de datos complejos?
Los objetos son considerados tipos de datos complejos porque pueden contener propiedades y métodos y permiten almacenar múltiples valores de diferentes tipos.

---

41. ¿Qué es una estructura de control en JavaScript?
Una estructura de control es una instrucción que permite alterar el flujo de ejecución de un programa, como condicionales o bucles.

---

42. ¿Cómo funciona la estructura if en JavaScript?
La estructura if ejecuta un bloque de código si se evalúa como verdadero la condición que le sigue.

---

43. ¿Qué es un else y cuándo se utiliza en JavaScript?
El else se utiliza junto con if para ejecutar un bloque de código alternativo cuando la condición de if es falsa.

---

44. ¿Qué hace el operador lógico && en una condición de JavaScript?
El operador && (Y lógico) devuelve true solo si ambas condiciones comparadas son verdaderas.

---

45. ¿Qué significa else if en una estructura condicional?
else if permite verificar múltiples condiciones secuenciales después de un if inicial, ejecutando el bloque correspondiente a la primera condición verdadera.

---

46. ¿Cómo se utiliza un switch en JavaScript?
El switch permite ejecutar diferentes bloques de código según el valor de una expresión, comparando este valor con casos definidos.

---

47. ¿Qué ocurre si no hay un break al final de un case en un switch?
Si no se utiliza break, la ejecución continúa en el siguiente case incluso si no cumple la condición, lo que puede llevar a un comportamiento inesperado.

---

48. ¿Qué es un bucle for en JavaScript?
El bucle for es una estructura de control que repite un bloque de código un número específico de veces, basado en una condición.

---

49. ¿Cómo se estructura un bucle for en JavaScript?
Un bucle for tiene la siguiente estructura: for (inicialización; condición; incremento/decremento) { // Código a ejecutar }.

---

50. ¿Qué hace el bucle while en JavaScript?
El bucle while repite un bloque de código mientras la condición especificada sea verdadera.

---

51. ¿Cuál es la diferencia entre un bucle for y un bucle while en JavaScript?
El bucle for se utiliza cuando se conoce cuántas veces se va a ejecutar el bloque, mientras que while se usa cuando la condición puede cambiar durante la ejecución.

---

52. ¿Qué es un bucle do...while y cómo se diferencia de while?
El bucle do...while ejecuta el bloque de código al menos

una vez antes de verificar la condición, mientras que en while la condición se evalúa antes de la primera ejecución.

---

53. ¿Qué es una declaración break en un bucle?
La declaración break termina la ejecución de un bucle o un switch de inmediato, incluso si la condición no se ha cumplido completamente.

---

54. ¿Qué hace la declaración continue en un bucle?
La declaración continue salta el resto del código dentro de un bucle y pasa a la siguiente iteración.

---

55. ¿Cómo funciona un bucle for...in en JavaScript?
El bucle for...in itera sobre las propiedades enumerables de un objeto, permitiendo acceder a las claves del objeto.

---

56. ¿Qué es un bucle for...of y en qué se diferencia de for...in?
El bucle for...of se utiliza para iterar sobre los valores de un array u objeto iterable, mientras que for...in itera sobre las claves de un objeto.

---

57. ¿Qué ocurre si no se encuentra una coincidencia en un switch en JavaScript?

Si no se encuentra una coincidencia en un switch, se ejecuta el bloque de código bajo default si está presente.

---

58. ¿Cómo puedes evitar que un bucle infinito ocurra en JavaScript?

Para evitar un bucle infinito, debes asegurarte de que la condición del bucle se modifique adecuadamente dentro del bloque de código para que eventualmente se vuelva falsa.

---

59. ¿Cómo se puede usar un switch para comparar cadenas en JavaScript?

Puedes usar switch para comparar cadenas colocando cada caso con el valor de la cadena y ejecutando el bloque correspondiente si hay una coincidencia.

---

60. ¿Qué sucede si la condición de un while es siempre false en JavaScript?

Si la condición de un while es siempre false, el bucle no ejecutará ninguna iteración y el código dentro de él será ignorado.

---

61. ¿Cómo se declara una función en JavaScript?
Una función en JavaScript se declara utilizando la palabra clave function, seguida de un nombre, paréntesis con parámetros y un bloque de código entre llaves:
function nombreFuncion(parametros) { // Código }.

---

62. ¿Qué es un parámetro en una función de JavaScript?
Un parámetro es una variable que se define en la declaración de la función y que se usa para recibir un valor cuando la función es llamada.

---

63. ¿Qué es un argumento en JavaScript?
Un argumento es el valor que se pasa a una función cuando es llamada, y corresponde a los parámetros definidos en la declaración de la función.

---

64. ¿Cómo se retorna un valor desde una función en JavaScript?
Se utiliza la palabra clave return seguida del valor que se desea devolver:
return valor;.

---

65. ¿Qué sucede si no se incluye un return en una función?

Si no se incluye un return, la función devolverá undefined de manera implícita.

---

66. ¿Qué es una función anónima en JavaScript?

Una función anónima es una función que no tiene un nombre y generalmente se asigna a una variable o se pasa como argumento a otra función.

---

67. ¿Cómo se define una función de flecha en JavaScript?

Una función de flecha se define utilizando la sintaxis () => {} en lugar de la palabra clave function:
const miFuncion = () => { // Código }.

---

68. ¿Cuál es la principal diferencia entre una función normal y una función de flecha en JavaScript?

La principal diferencia es que las funciones de flecha no tienen su propio this; en su lugar, heredan el valor de this del contexto en el que se definen.

---

69. ¿Qué significa que las funciones de flecha no tengan su propio this?
Significa que el valor de this dentro de una función de flecha es el mismo que el del contexto en el que la función fue creada, y no cambia como ocurre en las funciones normales.

---

70. ¿Qué es un closure en JavaScript?
Un closure es una función que tiene acceso a las variables de su propio ámbito, al ámbito de la función externa y al ámbito global, incluso después de que la función externa haya terminado de ejecutarse.

---

71. ¿Cómo se puede crear un closure en JavaScript?
Un closure se crea cuando una función interna accede a las variables de una función externa. Ejemplo:

```
function externa() {
  let variable = 'valor';
  return function interna() {
    console.log(variable);
  };
```

```
}
```

---

72. ¿Qué es la invocación inmediata de una función (IIFE)?

Una IIFE (Immediately Invoked Function Expression) es una función que se declara y se ejecuta inmediatamente, envolviéndola en paréntesis y agregando los paréntesis de llamada al final:
(function() { // Código })();.

---

73. ¿Qué es el paso por valor en una función de JavaScript?

El paso por valor ocurre cuando se pasan valores primitivos (como números o cadenas) a una función, y la función trabaja con una copia del valor original.

---

74. ¿Qué es el paso por referencia en una función de JavaScript?

El paso por referencia ocurre cuando se pasan objetos a una función, y la función trabaja con una referencia al objeto original.

---

75. ¿Cuál es la utilidad de los parámetros predeterminados en las funciones de JavaScript?
Los parámetros predeterminados permiten asignar un valor por defecto a los parámetros de una función en caso de que no se pase un valor al llamar la función.

---

76. ¿Cómo se definen parámetros predeterminados en una función?
Se definen al asignar un valor predeterminado en la declaración de la función:
function miFuncion(x = 5) { // Código }.

---

77. ¿Qué es una función recursiva en JavaScript?
Una función recursiva es una función que se llama a sí misma dentro de su propio cuerpo para resolver un problema dividiéndolo en subproblemas más pequeños.

---

78. ¿Qué es una función de orden superior en JavaScript?
Una función de orden superior es aquella que puede tomar otra función como argumento o devolver una función como resultado.

---

79. ¿Cómo se puede pasar una función como argumento a otra función en JavaScript?
Se puede pasar una función como argumento pasando su nombre sin los paréntesis, como por ejemplo: function llamarFuncion(func) { func(); }.

---

80. ¿Cuál es la diferencia entre var, let y const en el contexto de las funciones?

- var tiene un alcance de función y puede ser redeclarada.
- let tiene un alcance de bloque y no puede ser redeclarada dentro del mismo bloque.
- const tiene un alcance de bloque y no puede ser reasignada.

---

81. ¿Qué es un array en JavaScript?
Un array es una estructura de datos que permite almacenar múltiples valores en una sola variable, y los elementos pueden ser de diferentes tipos.

---

82. ¿Cómo se crea un array en JavaScript?
Un array se crea utilizando corchetes:
let miArray = [1, 2, 3];.

---

83. ¿Cómo acceder a un elemento de un array en JavaScript?
Se accede a un elemento de un array utilizando su índice, por ejemplo:
let elemento = miArray[0];.

---

84. ¿Qué hace el método map() en un array?
El método map() crea un nuevo array con los resultados de aplicar una función a cada elemento del array original.

---

85. ¿Cómo se utiliza el método filter() en un array?
El método filter() crea un nuevo array con los elementos que pasen una condición especificada en la función proporcionada.

---

86. ¿Qué hace el método reduce() en un array?
El método reduce() aplica una función acumuladora a

cada elemento de un array para reducirlo a un solo valor.

---

87. ¿Qué diferencia existe entre map() y filter() en JavaScript?
map() transforma todos los elementos de un array y devuelve un array con los nuevos elementos, mientras que filter() crea un nuevo array con solo los elementos que cumplen una condición.

---

88. ¿Qué hace el método forEach() en un array?
El método forEach() ejecuta una función en cada elemento de un array, pero no devuelve un nuevo array.

---

89. ¿Cómo se puede obtener el índice de un elemento en un array utilizando indexOf()?
El método indexOf() devuelve el primer índice en el que se encuentra un elemento en el array o -1 si no se encuentra:
let indice = miArray.indexOf(2);.

---

90. ¿Qué hace el método some() en un array?
El método some() verifica si al menos un elemento del

array cumple con la condición especificada y devuelve true o false.

---

91. ¿Qué hace el método every() en un array?
El método every() verifica si todos los elementos del array cumplen con la condición especificada y devuelve true si es así, o false en caso contrario.

---

92. ¿Cómo se puede combinar varios arrays en uno solo en JavaScript?
Se puede utilizar el método concat() o el operador de propagación (...) para combinar arrays:
let nuevoArray = array1.concat(array2); o let nuevoArray = [...array1, ...array2];.

---

93. ¿Qué hace el método find() en un array?
El método find() devuelve el primer elemento en el array que cumpla con la condición proporcionada en la función.

---

94. ¿Qué hace el método findIndex() en un array?
El método findIndex() devuelve el índice del primer

elemento en el array que cumpla con la condición proporcionada, o -1 si no se encuentra.

---

95. ¿Qué hace el método sort() en un array?
El método sort() ordena los elementos de un array en su lugar, convirtiéndolos en cadenas por defecto para la comparación.

---

96. ¿Cómo se puede invertir el orden de los elementos de un array en JavaScript?
Se puede utilizar el método reverse() para invertir el orden de los elementos en un array: miArray.reverse();.

---

97. ¿Qué hace el método splice() en un array?
El método splice() cambia el contenido de un array eliminando, reemplazando o añadiendo elementos en una posición específica.

---

98. ¿Qué hace el método slice() en un array?
El método slice() devuelve una copia superficial de una porción de un array, sin modificar el array original.

99. ¿Qué hace el método push() en un array?
El método push() añade uno o más elementos al final de un array y devuelve la nueva longitud del array.

---

100. ¿Qué hace el método pop() en un array?
El método pop() elimina el último elemento de un array y lo devuelve.

101. ¿Qué es el DOM (Document Object Model) en JavaScript?
El DOM es una interfaz de programación que representa la estructura de un documento HTML como un árbol de nodos, donde cada nodo es un objeto que corresponde a una parte del documento.

---

102. ¿Cómo puedes seleccionar un elemento por su ID en el DOM?
Se puede seleccionar un elemento por su ID utilizando el método getElementById(), por ejemplo:
let elemento = document.getElementById('miId');.

---

103. ¿Qué es querySelector() en JavaScript?
querySelector() es un método que permite seleccionar el primer elemento que coincide con un selector CSS en el DOM, como una clase, ID o etiqueta:
let elemento = document.querySelector('.miClase');.

---

104. ¿Cuál es la diferencia entre getElementById() y querySelector()?
getElementById() selecciona un único elemento con un ID específico, mientras que querySelector() puede seleccionar cualquier elemento que coincida con un selector CSS (ID, clase, etiqueta, etc.).

---

105. ¿Cómo se seleccionan todos los elementos que coinciden con un selector CSS?
Se puede usar el método querySelectorAll(), que devuelve una lista estática de todos los elementos que coinciden con el selector:
let elementos = document.querySelectorAll('.miClase');.

---

106. ¿Qué es getElementsByClassName() en JavaScript?
getElementsByClassName() es un método que selecciona todos los elementos que tienen una clase específica, devolviendo una colección de nodos.

---

107. ¿Qué es getElementsByTagName() en JavaScript?
getElementsByTagName() es un método que selecciona todos los elementos de un tipo de etiqueta específica (como , <p>, etc.), devolviendo una colección de nodos.

---

108. ¿Cómo puedes crear un nuevo elemento en el DOM?
Se utiliza el método document.createElement(), pasando el nombre de la etiqueta que deseas crear:
let nuevoElemento = document.createElement('div');.

---

109. ¿Cómo se puede añadir un nuevo elemento al DOM?
Se utiliza el método appendChild() para agregar un nuevo nodo al final de un contenedor:
parentNode.appendChild(nuevoElemento);.

---

110. ¿Cómo puedes insertar un nuevo elemento antes de otro existente en el DOM?
Se utiliza el método insertBefore(), que permite insertar un nuevo nodo antes de un nodo de referencia específico:

parentNode.insertBefore(nuevoElemento, nodoReferencia);.

---

111. ¿Cómo puedes eliminar un elemento del DOM?
Para eliminar un elemento, se usa el método removeChild() del nodo padre: parentNode.removeChild(elemento);.

---

112. ¿Cómo puedes modificar el contenido de un elemento en el DOM?
Puedes modificar el contenido de un elemento utilizando la propiedad innerHTML o textContent: elemento.innerHTML = 'Nuevo contenido';.

---

113. ¿Cómo se cambia el estilo de un elemento en el DOM?
Para cambiar el estilo de un elemento, se puede acceder a la propiedad style del mismo: elemento.style.color = 'red';.

---

114. ¿Cómo se añaden o eliminan clases a un elemento en el DOM?
Se utiliza la propiedad classList, que ofrece métodos

como add(), remove(), y toggle(): elemento.classList.add('nuevaClase');.

---

115. ¿Qué es createTextNode() en el DOM?
createTextNode() es un método que crea un nodo de texto que puede ser añadido a un elemento en el DOM: let nodoTexto = document.createTextNode('Texto aquí');.

---

116. ¿Cómo puedes cambiar el atributo src de una imagen en el DOM?
Se puede cambiar el atributo src de una imagen accediendo al elemento y modificando el atributo: imagen.setAttribute('src', 'nuevaImagen.jpg');.

117. ¿Cómo puedes acceder a los atributos de un elemento en el DOM?
Se puede acceder a los atributos de un elemento utilizando el método getAttribute(), por ejemplo: let src = imagen.getAttribute('src');.

---

118. ¿Qué es un evento en JavaScript?
Un evento en JavaScript es una acción o suceso que ocurre en el navegador, como un clic, un

desplazamiento, una pulsación de tecla, entre otros, que puede ser detectado y manipulado mediante el código.

---

119. ¿Cómo se asocia un evento a un elemento en JavaScript?

Un evento se asocia a un elemento utilizando el método addEventListener(), donde se especifica el tipo de evento y la función que debe ejecutarse cuando el evento se dispare:
elemento.addEventListener('click', funcion);.

---

120. ¿Qué es el modelo de propagación de eventos en JavaScript?

El modelo de propagación de eventos describe cómo un evento se mueve a través de los nodos del DOM. Existen dos fases: la fase de captura y la fase de burbujeo.

---

121. ¿Qué es la fase de captura en el modelo de propagación de eventos?

La fase de captura es la fase inicial en la que el evento se propaga desde el nodo raíz del DOM hasta el objetivo del evento antes de llegar al elemento que lo disparó.

---

122. ¿Qué es la fase de burbujeo en el modelo de propagación de eventos?
La fase de burbujeo ocurre después de que el evento ha alcanzado el elemento objetivo y comienza a propagarse hacia arriba a través del árbol del DOM hasta el nodo raíz.

---

123. ¿Cómo se puede evitar la propagación de un evento en JavaScript?
Se puede evitar que un evento se propague utilizando el método stopPropagation() dentro del manejador del evento:
evento.stopPropagation();.

---

124. ¿Cómo se puede prevenir el comportamiento por defecto de un evento en JavaScript?
Para evitar el comportamiento predeterminado de un evento, como la acción de un enlace, se utiliza el método preventDefault():
evento.preventDefault();.

---

125. ¿Cómo se maneja un evento de clic (click) en JavaScript?
Un evento de clic se maneja utilizando addEventListener(), especificando el tipo de evento como

'click' y la función a ejecutar: elemento.addEventListener('click', funcion);.

---

126. ¿Qué es el evento mouseover en JavaScript?
El evento mouseover se activa cuando el puntero del ratón se desplaza sobre un elemento, y se puede usar para activar efectos visuales, como cambios en el estilo.

---

127. ¿Cómo se maneja un evento mouseover en JavaScript?
Un evento mouseover se maneja de manera similar a otros eventos, utilizando addEventListener() para asociar la función al evento: elemento.addEventListener('mouseover', funcion);.

---

128. ¿Qué es el evento mouseout en JavaScript?
El evento mouseout se activa cuando el puntero del ratón sale de un elemento y se utiliza comúnmente para revertir efectos visuales.

---

129. ¿Cómo se maneja un evento mouseout en JavaScript?
Un evento mouseout se maneja asociando una función

al evento con addEventListener(): elemento.addEventListener('mouseout', funcion);.

---

130. ¿Qué es el evento keydown en JavaScript?
El evento keydown se activa cuando una tecla es presionada, y puede ser utilizado para manejar la entrada de texto o comandos del teclado.

---

131. ¿Cómo se maneja un evento keydown en JavaScript?
Un evento keydown se maneja de la misma manera que otros eventos, utilizando addEventListener(): document.addEventListener('keydown', funcion);.

---

132. ¿Qué es el evento keyup en JavaScript?
El evento keyup se activa cuando se suelta una tecla previamente presionada y se puede usar para realizar acciones una vez que la tecla ha dejado de ser presionada.

---

133. ¿Cómo se maneja un evento keyup en JavaScript?
Se maneja con addEventListener(), especificando el tipo

de evento como 'keyup' y la función a ejecutar: document.addEventListener('keyup', funcion);.

---

134. ¿Cómo se puede manejar el evento de doble clic (dblclick) en JavaScript?
Un evento de doble clic se maneja de forma similar a los demás eventos, utilizando addEventListener() con el tipo de evento 'dblclick': elemento.addEventListener('dblclick', funcion);.

---

135. ¿Qué es el evento focus en JavaScript?
El evento focus se dispara cuando un elemento (como un campo de entrada de texto) recibe el foco, es decir, cuando el usuario hace clic en él o lo selecciona con el teclado.

---

136. ¿Cómo se maneja un evento focus en JavaScript?
Se maneja utilizando addEventListener(), especificando el evento 'focus': elemento.addEventListener('focus', funcion);.

---

137. ¿Qué es el evento blur en JavaScript?
El evento blur se activa cuando un elemento pierde el

foco, es decir, cuando el usuario hace clic fuera del elemento o cambia el enfoque a otro elemento.

---

138. ¿Cómo se maneja un evento blur en JavaScript?
Se maneja de forma similar a otros eventos, utilizando addEventListener() con el tipo de evento 'blur': elemento.addEventListener('blur', funcion);.

---

139. ¿Cómo se puede asociar varios eventos a un mismo elemento en JavaScript?
Puedes asociar varios eventos a un mismo elemento utilizando addEventListener() varias veces para cada tipo de evento:
elemento.addEventListener('click', funcion1);
elemento.addEventListener('mouseover', funcion2);.

---

140. ¿Qué es el evento change en JavaScript?
El evento change se activa cuando el valor de un campo de formulario, como un campo de entrada o un selector, cambia.

---

141. ¿Cómo se maneja un evento change en JavaScript?

Se maneja utilizando addEventListener(), especificando el tipo de evento como 'change': elemento.addEventListener('change', funcion);

---

142. ¿Qué es la asincronía en JavaScript?
La asincronía en JavaScript permite que el código se ejecute de manera no bloqueante, lo que significa que las operaciones como la lectura de archivos o solicitudes HTTP pueden ejecutarse mientras el resto del código continúa funcionando.

---

143. ¿Qué es un callback en JavaScript?
Un callback es una función que se pasa como argumento a otra función y que se ejecuta cuando la operación asincrónica ha terminado.

---

144. ¿Cómo funciona un callback en una operación asíncrona?
El callback se pasa como argumento a una función asíncrona y se ejecuta cuando la operación se completa, permitiendo continuar con el flujo de ejecución.

---

145. ¿Qué es el "callback hell" en JavaScript?
El "callback hell" es el término utilizado para describir una situación donde las funciones de callback están anidadas varias veces, lo que dificulta la lectura y mantenimiento del código.

---

146. ¿Qué son las promesas en JavaScript?
Las promesas son objetos que representan el eventual resultado de una operación asincrónica, pudiendo estar en uno de tres estados: pendiente, resuelta o rechazada.

---

147. ¿Cómo se crea una promesa en JavaScript?
Una promesa se crea utilizando el constructor Promise(), pasando una función que tiene dos parámetros: resolve y reject.
let promesa = new Promise((resolve, reject) => { /* código */ });.

---

148. ¿Cómo se maneja una promesa en JavaScript?
Las promesas se manejan con los métodos then() y catch(), donde then() maneja el caso de éxito y catch() maneja el caso de error:
promesa.then(resultado => { /* éxito */ }).catch(error => { /* error */ });.

---

149. ¿Qué es el método finally() de una promesa?
El método finally() de una promesa se ejecuta después de que la promesa se haya resuelto o rechazado, independientemente de su resultado, y se usa para realizar una acción final, como la limpieza.

---

150. ¿Qué es async/await en JavaScript?
async/await es una sintaxis que simplifica el trabajo con promesas, permitiendo escribir código asíncrono de manera más legible, parecida a un código sincrónico.

---

151. ¿Cómo se declara una función asíncrona en JavaScript?
Una función asíncrona se declara con la palabra clave async, lo que permite el uso de await dentro de la función para esperar a que las promesas se resuelvan: async function miFuncion() { /* código */ }.

---

152. ¿Qué hace el operador await en JavaScript?
El operador await se utiliza dentro de una función async para esperar a que una promesa se resuelva o se rechace antes de continuar con la ejecución del código.

153. ¿Cómo manejar errores con async/await en JavaScript?

Los errores con async/await se pueden manejar utilizando un bloque try/catch, donde se intenta ejecutar el código asíncrono y se captura cualquier error que ocurra:
try { /* código */ } catch (error) { /* manejar error */ }.

---

154. ¿Qué es la Fetch API en JavaScript?

La Fetch API es una interfaz que permite realizar solicitudes HTTP asíncronas en JavaScript para obtener o enviar datos a un servidor, y es más moderna que XMLHttpRequest.

---

155. ¿Cómo realizar una solicitud GET usando la Fetch API?

Se realiza una solicitud GET con la Fetch API utilizando el siguiente código:
fetch('url').then(response => response.json()).then(data => { /* manejar datos */ }).catch(error => { /* manejar error */ });.

---

156. ¿Qué es el método json() en la Fetch API?
El método json() convierte una respuesta HTTP en formato JSON a un objeto JavaScript, y se utiliza después de obtener la respuesta con fetch().

---

157. ¿Cómo enviar datos en una solicitud POST usando la Fetch API?
Se pueden enviar datos en una solicitud POST configurando el método y los encabezados apropiados: fetch('url', { method: 'POST', body: JSON.stringify(datos), headers: { 'Content-Type': 'application/json' } });.

---

158. ¿Qué es la respuesta response.ok en la Fetch API?
La propiedad response.ok es un valor booleano que indica si la respuesta HTTP fue exitosa (estado 2xx) o no.

---

159. ¿Cómo manejar errores de red con la Fetch API?
Los errores de red en la Fetch API se manejan utilizando catch() después de realizar la solicitud: fetch('url').catch(error => { /* manejar error */ });.

---

160. ¿Cómo convertir una respuesta de la Fetch API en texto en lugar de JSON?

Se puede convertir una respuesta en texto utilizando el método text() de la respuesta:
fetch('url').then(response => response.text()).then(text => { /* manejar texto */ });.

---

161. ¿Cómo hacer que una solicitud Fetch sea asincrónica utilizando async/await?

Para hacer que una solicitud Fetch sea asincrónica, se utiliza await dentro de una función async:
let respuesta = await fetch('url');.

---

162. ¿Qué es la propiedad response.status en la Fetch API?

La propiedad response.status devuelve el código de estado HTTP de la respuesta, como 200 para una solicitud exitosa o 404 para "no encontrado".

---

163. ¿Cómo configurar los encabezados en una solicitud Fetch?

Se pueden configurar los encabezados de una solicitud Fetch utilizando la propiedad headers dentro de las opciones de la solicitud:
fetch('url', { headers: { 'Authorization': 'Bearer token' } });.

---

164. ¿Qué es la programación orientada a objetos en JavaScript?

La programación orientada a objetos (POO) en JavaScript es un paradigma que organiza el código en objetos que contienen tanto datos como métodos para operar sobre esos datos.

---

165. ¿Qué es un prototipo en JavaScript?

Un prototipo es un objeto del que otro objeto hereda propiedades y métodos. JavaScript usa herencia prototípica para compartir comportamientos entre objetos.

---

166. ¿Cómo se crea un objeto con un prototipo en JavaScript?

Un objeto puede crear un prototipo usando Object.create() o estableciendo el prototipo directamente mediante prototype.
let obj = Object.create(prototipo);.

---

167. ¿Qué es la herencia prototípica en JavaScript?

La herencia prototípica es el mecanismo por el cual los

objetos en JavaScript pueden heredar propiedades y métodos de otros objetos a través de su prototipo.

---

168. ¿Cómo puedes acceder al prototipo de un objeto en JavaScript?
Se puede acceder al prototipo de un objeto utilizando Object.getPrototypeOf(), o accediendo a la propiedad __proto__.
let proto = Object.getPrototypeOf(obj);.

---

169. ¿Qué es la propiedad constructor en JavaScript?
La propiedad constructor es una referencia a la función constructora que creó un objeto, y está presente en todos los objetos que heredan de un prototipo.

---

170. ¿Cómo se define una clase en JavaScript?
Una clase en JavaScript se define utilizando la palabra clave class, seguida del nombre de la clase y un bloque de código con un constructor y métodos:
class Persona { constructor(nombre) { this.nombre = nombre; } }.

---

171. ¿Qué es un constructor en una clase en JavaScript?
El constructor es un método especial dentro de una clase que se invoca al crear una nueva instancia de la clase, y se usa para inicializar las propiedades del objeto.

---

172. ¿Cómo se crea una instancia de una clase en JavaScript?
Se crea una instancia de una clase utilizando la palabra clave new, seguida del nombre de la clase:
let persona = new Persona('Juan');.

---

173. ¿Qué es la herencia en clases de JavaScript?
La herencia en clases permite que una clase hija herede propiedades y métodos de una clase padre, utilizando la palabra clave extends.
class Estudiante extends Persona { /* métodos adicionales */ }.

---

174. ¿Qué es el polimorfismo en JavaScript?
El polimorfismo es un principio de la POO donde un mismo método puede tener diferentes implementaciones dependiendo del tipo de objeto que lo invoque.

---

175. ¿Cómo se invoca el método del padre en una clase hija en JavaScript?

Se puede invocar el método del padre utilizando super(), que permite acceder a las propiedades y métodos de la clase base.
super.metodo();.

---

176. ¿Cómo se pueden definir métodos dentro de una clase en JavaScript?

Los métodos dentro de una clase se definen como funciones dentro del cuerpo de la clase, sin necesidad de la palabra clave function:
class Persona { saludar() { console.log('Hola'); } }.

---

177. ¿Qué es la palabra clave this en JavaScript?

La palabra clave this se refiere al contexto de ejecución del código y hace referencia al objeto sobre el que se está invocando el método o función.

---

178. ¿Cómo cambia el valor de this en función del contexto de ejecución?

El valor de this cambia dependiendo de cómo se invoque una función. En métodos de objetos, this se refiere al

objeto; en funciones globales, this hace referencia al objeto global.

---

179. ¿Cómo se puede cambiar el valor de this en JavaScript?
El valor de this puede cambiar mediante los métodos call(), apply() y bind(), que permiten explícitamente especificar el valor de this.
miFuncion.call(objeto);.

---

180. ¿Cómo se utiliza super en una clase hija?
La palabra clave super se usa en una clase hija para llamar al constructor o métodos de la clase padre. Se invoca en el constructor o en un método:
super(); en el constructor o super.metodo(); en un método.

---

181. ¿Qué es la encapsulación en JavaScript?
La encapsulación es un principio de la POO que restringe el acceso directo a algunas de las propiedades y métodos de un objeto, proporcionando una interfaz pública para interactuar con ellos.

---

182. ¿Cómo se simula la encapsulación en JavaScript?
La encapsulación en JavaScript se puede simular utilizando propiedades privadas (utilizando símbolos o métodos), y exponiendo solo los métodos necesarios mediante la creación de getters y setters.

---

183. ¿Qué es un getter y un setter en JavaScript?
Un getter es un método que permite acceder al valor de una propiedad, mientras que un setter permite modificar el valor de una propiedad. Se definen con get y set en la clase.

---

184. ¿Cómo se implementan getters y setters en JavaScript?
Los getters y setters se implementan dentro de una clase con la palabra clave get para obtener el valor y set para establecer el valor:
get propiedad() { return this._propiedad; }
set propiedad(valor) { this._propiedad = valor; }.

---

185. ¿Cuál es la diferencia entre Object.create() y una clase en JavaScript?
Object.create() crea un nuevo objeto con un prototipo específico, mientras que una clase en JavaScript es una

plantilla para crear objetos con un constructor y métodos, utilizando la sintaxis moderna class.

---

186. ¿Cuál es la diferencia entre let y const en cuanto a su alcance?

Ambas variables tienen alcance de bloque, lo que significa que están limitadas al bloque en el que se definen, pero const no puede ser reasignado, mientras que let sí.

---

187. ¿Qué significa la inmutabilidad en el contexto de const?

La inmutabilidad de const significa que el valor de la variable no puede ser reasignado, pero si se trata de un objeto o array, sus propiedades o elementos pueden modificarse.

---

188. ¿Qué son las funciones de flecha (arrow functions) en JavaScript?

Las funciones de flecha son una forma más concisa de declarar funciones, usando la sintaxis () => {} y no tienen su propio this, lo que las hace útiles para preservar el contexto de ejecución.

---

189. ¿Cuál es la diferencia en el uso de this en funciones tradicionales y funciones de flecha?
Las funciones tradicionales definen su propio this, mientras que las funciones de flecha heredan el this del contexto donde fueron creadas, sin crear uno nuevo.

---

190. ¿Cómo se define una clase en ES6?
Se utiliza la palabra clave class para definir una clase en ES6. Dentro de la clase se puede definir un constructor y métodos:
class Persona { constructor(nombre) { this.nombre = nombre; } }.

---

191. ¿Cómo se crea un objeto a partir de una clase en ES6?
Un objeto se crea utilizando la palabra clave new seguida del nombre de la clase:
let persona = new Persona('Juan');.

---

192. ¿Qué son los módulos en ES6?
Los módulos en ES6 permiten dividir el código en archivos separados, facilitando la importación y exportación de funcionalidades entre diferentes partes de la aplicación.

---

193. ¿Cómo se importa una función desde otro módulo en ES6?

Se importa utilizando la palabra clave import:
import { miFuncion } from './miModulo.js';.

---

194. ¿Cómo se exporta una función desde un módulo en ES6?

Se exporta utilizando la palabra clave export:
export function miFuncion() { /* código */ }.

---

195. ¿Qué significa la desestructuración de objetos en ES6?

La desestructuración de objetos permite extraer propiedades de un objeto y asignarlas a variables de manera más concisa:
const { nombre, edad } = persona;.

---

196. ¿Cómo se hace la desestructuración de un array en ES6?

Se puede hacer la desestructuración de un array asignando elementos a variables de forma directa:
const [a, b] = [1, 2];.

---

197. ¿Qué es el spread operator (...) en JavaScript?
El spread operator se utiliza para expandir elementos de un array u objeto en nuevos arrays u objetos.
const nuevoArray = [...array1, ...array2];.

---

198. ¿Qué es el rest operator (...) en JavaScript?
El rest operator se utiliza para agrupar elementos de un array u objeto en una variable, recogiendo todos los valores restantes:
function suma(...numeros) { /* código */ }.

---

199. ¿Cómo se utiliza el spread operator para clonar un objeto?
El spread operator se puede usar para crear una copia superficial de un objeto:
const nuevoObjeto = { ...objetoOriginal };.

---

200. ¿Qué ventajas ofrece la desestructuración de objetos en comparación con la forma tradicional?
La desestructuración permite extraer valores de manera más clara y concisa, reduciendo la cantidad de código necesario y mejorando la legibilidad.

201. ¿Qué es la asignación por desestructuración en arrays?

La asignación por desestructuración en arrays permite asignar elementos del array a variables con la misma sintaxis que los objetos, simplificando la extracción de datos:
const [primero, segundo] = array;.

---

202. ¿Cómo utilizar el spread operator para combinar objetos?

El spread operator permite combinar varios objetos en uno nuevo, copiando todas sus propiedades:
const combinado = { ...objeto1, ...objeto2 };.

---

203. ¿Qué es un método getter y setter en el contexto de clases de ES6?

Los métodos getter y setter permiten acceder y modificar propiedades de una clase de forma controlada. Se definen con las palabras clave get y set.
get propiedad() { return this._propiedad; }
set propiedad(valor) { this._propiedad = valor; }.

---

204. ¿Qué son los métodos estáticos en clases de ES6?
Los métodos estáticos son métodos que pertenecen a la clase en sí, no a las instancias de la clase. Se definen utilizando la palabra clave static:
static metodoEstatico() { /* código */ }.

---

205. ¿Qué es el módulo default en ES6?
Un módulo default permite exportar un único valor o función como la exportación principal del archivo, lo que facilita su importación sin necesidad de llaves:
export default miFuncion;.

---

206. ¿Qué es un closure en JavaScript y cómo funciona?
Un closure es una función que retiene acceso a su contexto léxico, incluso después de que la función exterior haya terminado de ejecutarse. Esto permite que la función interna acceda a las variables de la función externa.

---

207. ¿Qué son los patrones de diseño y cómo se aplican en JavaScript?
Los patrones de diseño son soluciones reutilizables a problemas comunes en el desarrollo de software. En JavaScript, patrones como el módulo, observador,

singleton y factory se utilizan para organizar el código, mejorar la reutilización y facilitar el mantenimiento.

Y esto es todo.

Si quieres seguir aprendiendo JavaScript puedes hacerlo aquí:

https://victorroblesweb.es/master-javascript-moderno

Y si quieres aprender desarrollo web desde cero hasta estar preparado para trabajar al menos como junior, te recomiendo que sigas esta ruta de aprendizaje y programes mucho:

https://victorroblesweb.es/ruta

Gracias,

Nos vemos y nos leemos

Atentamente, Víctor Robles WEB ;)