

Preguntas 1er Parcial - TUP 2025 - P4

1. ¿Cuál de estos es un identificador **válido**?

- `_variable`
 - `var-iable`
 - `2variable`
-

2. ¿Cuál de estos identificadores es **inválido**?

- `$total`
 - `for`
 - `nombreUsuario`
-

3. ¿Qué identificador es considerado **válido**?

- `function`
 - `PI_MAX`
 - `const`
-

4. ¿Cuál de estos nombres de variable es **incorrecto**?

- `_resultadoFinal`
 - `dato123`
 - `123dato`
-

5. ¿Qué identificador es **válido**?

- `class`
 - `precio-total`
 - `precio$`
-

6. ¿Cuál es el identificador **aceptado** por JavaScript?

- NombreCompleto
 - Nombre-Completo
 - let
-

7. ¿Cuál de estos nombres de variable es **inválido**?

- \$temp
 - var
 - _temp
-

8. ¿Qué identificador es **válido**?

- else
 - if
 - mi_variable
-

9. ¿Cuál de estos identificadores puede usarse sin error?

- \$datoFinal
 - new
 - try
-

10. ¿Qué nombre de variable es **incorrecto**?

- usuario_final
 - usuario1
 - 1usuario
-

11. ¿Cuál declara una variable `x` con el valor numérico `10`?

-

```
let x = "10";
```

```
let x = 10;
```

```
let x = Number;
```

12. ¿Cuál de estas asignaciones genera el valor especial `NaN` ?

```
let n = Number("abc");
```

```
let n = +"42";
```

```
let n = 123;
```

13. ¿Cuál es una declaración válida de número decimal?

```
let pi = Number;
```

```
let pi = 3.14;
```

```
let pi = "3.14";
```

14. ¿Qué valor tendrá la variable `n`?

```
let n = +"5";
```

- 5
 - NaN
 - "5"
-

15. ¿Cuál de estas es la forma correcta de convertir un `string` a número?

-

```
let n = number("42");
```

-

```
let n = Number("42");
```

-

```
let n = Num("42");
```

16. ¿Qué declaración es válida para una constante numérica?

-

```
const pi = 3.1416;
```

-

```
const pi; pi = 3.1416;
```

```
const pi;
```

17. ¿Cuál inicializa la variable `n` con `Infinity`?

```
let n = "Infinity";
```

```
let n = infinite;
```

```
let n = Infinity;
```

18. ¿Qué resultado obtiene la variable `n`?

```
let n = parseInt("123abc");
```

- "123abc"
- 123
- NaN

19. ¿Cuál de estas declaraciones numéricas es **inválida**?

```
let x = 0x1F;
```

```
let x = 0b1010;
```

```
let x = 0y77;
```

20. ¿Qué valor tiene la variable `n`?

```
let n = parseFloat("3.14abc");
```

- 3.14
 - NaN
 - "3.14abc"
-

21. ¿Qué valor imprime el siguiente código?

```
console.log(2 + 3 * 4);
```

- 20
 - 14
 - 24
-

22. ¿Qué resultado produce esta operación?

```
console.log(10 - 4 / 2);
```

- 8
 - 3
 - 5
-

23. ¿Cuál es el resultado de la siguiente expresión?

```
console.log((2 + 3) * 4);
```

- 14
 - 20
 - 24
-

24. ¿Qué resultado obtiene `x`?

```
let x = 2 ** 3 * 2;  
console.log(x);
```

- 64
 - 256
 - 16
-

25. ¿Qué imprime el siguiente código?

```
console.log(10 / 2 * 5);
```

- 25
 - 1
 - 2.5
-

26. ¿Cuál es el valor de la expresión?

```
console.log(5 + 10 % 3);
```

- 6
 - 7
 - 8
-

27. ¿Qué resultado se obtiene?

```
console.log(2 + 3 * 2 ** 2);
```

- 20
 - 22
 - 14
-

28. ¿Qué valor se imprime?

```
console.log(100 / (5 * 2));
```

- 25
 - 20
 - 10
-

29. ¿Cuál es el resultado?

```
console.log(10 - 2 + 3);
```

- 11
 - 15
 - 5
-

30. ¿Qué valor tiene la variable `y`?

```
let y = 2 * 3 ** 2;  
console.log(y);
```

- 18
 - 36
 - 12
-

31. ¿Cuál es una declaración válida de un `string`?

```
let s = Hola;
```

```
let s = `Hola;
```

```
let s = 'Hola';
```

32. ¿Qué forma correcta declara un `string` con comillas dobles?

```
let texto = "Hola mundo";
```

```
let texto = 'Hola mundo';
```

```
let texto = Hola mundo;
```

33. ¿Cuál de estas formas es correcta para usar **plantillas literales**?

```
let saludo = `Hola ${nombre}`;
```

```
let saludo = "Hola ${nombre}";
```

```
let saludo = 'Hola ${nombre}';
```

34. ¿Qué declaración es válida?

```
let mensaje = It's ok;
```

```
let mensaje = 'It's ok';
```

```
let mensaje = "It's ok";
```

35. ¿Cuál de estas es la forma correcta de declarar un `string` vacío?

```
let s = "";
```

```
let s = ;
```

```
let s = ' ';
```

36. ¿Cuál declaración es válida con comillas simples?

```
let nombre = 'Juan';
```

```
let nombre = Juan;
```

```
let nombre = `Juan;
```

37. ¿Qué imprime este código?

```
let s = "Hola";
console.log(typeof s);
```

- "text"
 - "char"
 - "string"
-

38. ¿Qué declaración es correcta para un `string` multilínea?

```
let texto = `Primera
Segunda`;
```

```
let texto = 'Primera
Segunda';
```

```
let texto = "Primera
Segunda";
```

39. ¿Cuál de estas es la forma correcta de incluir comillas dobles dentro de un `string` ?

```
let s = 'Ella dijo: "Hola"';
```

```
let s = "Ella dijo: "Hola"";
```

```
let s = "Ella dijo: "Hola"";
```

40. ¿Qué resultado da este código?

```
let saludo = `Hola`;
console.log(saludo);
```

-
- `Hola`
 - `"Hola"`
 - `undefined`

41. ¿Qué resultado imprime el siguiente código?

```
console.log("Hola" + " Mundo");
```

-
- `"Hola Mundo"`
 - `"HolaMundo"`
 - `"Hola + Mundo"`

42. ¿Cuál es el resultado?

```
console.log("5" + 3);
```

- "53"
 - 8
 - "5 + 3"
-

43. ¿Qué imprime este código?

```
console.log("10" - 2);
```

- 8
 - "NaN"
 - "102"
-

44. ¿Qué valor devuelve?

```
console.log("5" * "2");
```

- "52"
 - 10
 - NaN
-

45. ¿Qué resultado se obtiene?

```
console.log("Hola".length + " Mundo".length);
```

- "HolaMundo"
 - 10
 - "10"
-

46. ¿Qué devuelve el siguiente código?

```
console.log("Hola".toUpperCase());
```

- "hola"

- "Hola"
 - "HOLA"
-

47. ¿Qué imprime este código?

```
console.log("HOLA".toLowerCase());
```

- "HOLA"
 - "Hola"
 - "hola"
-

48. ¿Cuál es el resultado?

```
console.log("JavaScript".charAt(4));
```

- "a"
 - "S"
 - "s"
-

49. ¿Qué valor devuelve?

```
console.log("Hola".includes("Ho"));
```

- true
 - false
 - "Ho"
-

50. ¿Qué imprime este código?

```
console.log("Hola Mundo".indexOf("Mundo"));
```

- 1
- 5

51. ¿Qué resultado devuelve?

```
console.log("Hola Mundo".slice(0, 4));
```

- "Mundo"
 - "ola"
 - "Hola"
-

52. ¿Qué devuelve este código?

```
console.log("Hola".repeat(3));
```

- "3Hola"
 - "HolaHolaHola"
 - "Hola3"
-

53. ¿Qué imprime este código?

```
console.log(" Hola ".trim());
```

- " Hola "
 - "Hola "
 - "Hola"
-

54. ¿Qué devuelve?

```
console.log("Hola".startsWith("Ho"));
```

- true
- false
- "Ho"

55. ¿Qué resultado se obtiene?

```
console.log("Hola".endsWith("la"));
```

- false
- "la"
- true

56. ¿Cuál es la forma correcta de declarar un valor booleano true ?

-

```
let activo = true;
```

-

```
let activo = True;
```

-

```
let activo = "true";
```

57. ¿Cuál de las siguientes asignaciones representa un valor booleano false ?

-

```
let valido = false;
```

-

```
let valido = "false";
```

-

```
let valido = 0;
```

58. ¿Qué valor tendrá la variable `flag` ?

```
let flag = Boolean(0);
```

- `true`
- `false`
- `"false"`

59. ¿Cuál es el resultado de esta declaración?

```
let estado = Boolean("Hola");
```

- `false`
- `"true"`
- `true`

60. ¿Qué imprime el siguiente código?

```
console.log(typeof false);
```

- `"string"`
- `"boolean"`
- `"false"`

61. ¿Cuál es el resultado de la conversión?

```
console.log(Boolean(""));
```

- `true`
- `false`
- `"false"`

62. ¿Qué valor imprime?

```
console.log(Boolean(0));
```

- true
 - false
 - "0"
-

63. ¿Cuál será el resultado?

```
console.log(Boolean("Hola"));
```

- false
 - true
 - "true"
-

64. ¿Qué valor devuelve este código?

```
console.log(Boolean(null));
```

- true
 - false
 - "null"
-

65. ¿Qué resultado se obtiene?

```
console.log("") || "Texto";
```

- ""
 - false
 - "Texto"
-

66. ¿Cuál será la salida?

```
console.log("Hola" && "Mundo");
```

- "Mundo"
 - true
 - "Hola"
-

67. ¿Qué valor imprime?

```
console.log(0 || 123);
```

- 0
 - 123
 - false
-

68. ¿Cuál es el resultado de esta operación?

```
console.log(0 && "Texto");
```

- 0
 - false
 - "Texto"
-

69. ¿Qué imprime el siguiente código?

```
console.log(null || undefined || "valor");
```

- undefined
 - null
 - "valor"
-

70. ¿Qué resultado se obtiene?

```
console.log("Texto" && 0 && true);
```

- true
 - "Texto"
 - 0
-

71. ¿Cuál es el valor de una variable declarada pero no inicializada?

```
let x;  
console.log(x);
```

- undefined
 - null
 - 0
-

72. ¿Qué imprime el siguiente código?

```
let y = null;  
console.log(y);
```

- false
 - null
 - undefined
-

73. ¿Qué tipo devuelve `typeof undefined` ?

- "object"
 - "null"
 - "undefined"
-

74. ¿Qué tipo devuelve `typeof null` ?

- "null"
- "undefined"

"object"

75. ¿Qué valor imprime este código?

```
let a = null;  
let b;  
console.log(a === b);
```

false
 "false"
 true

76. ¿Cuál es el resultado de la comparación?

```
console.log(null == undefined);
```

true
 NaN
 false

77. ¿Qué imprime este código?

```
console.log(null === undefined);
```

NaN
 true
 false

78. ¿Qué valor tendrá z ?

```
let z = undefined || "valor";  
console.log(z);
```

null

- undefined
 - "valor"
-

79. ¿Cuál es el resultado de esta operación?

```
let x = null ?? "alternativo";
console.log(x);
```

- null
 - undefined
 - "alternativo"
-

80. ¿Qué imprime este código?

```
let y = undefined ?? null;
console.log(y);
```

- null
 - "undefined"
 - undefined
-

81. ¿Qué resultado imprime este código?

```
console.log(true && false);
```

- "false"
 - true
 - false
-

82. ¿Qué valor devuelve?

```
console.log(true || false);
```

- "true"

- `false`
 - `true`
-

83. ¿Qué imprime el siguiente código?

```
console.log(!true);
```

- `"false"`
 - `false`
 - `true`
-

84. ¿Cuál será el resultado?

```
console.log(false || (true && false));
```

- `false`
 - `true`
 - `undefined`
-

85. ¿Qué valor imprime?

```
console.log((true && true) || false);
```

- `false`
 - `true`
 - `"true"`
-

86. ¿Qué devuelve este código?

```
console.log(!(false || true));
```

- `true`
- `false`

"false"

87. ¿Cuál es el resultado de la operación?

```
console.log(true && "Texto");
```

"Texto"
 false
 true

88. ¿Qué imprime este código?

```
console.log(false && "Hola");
```

false
 "Hola"
 undefined

89. ¿Qué valor devuelve?

```
console.log(null || true);
```

true
 null
 false

90. ¿Cuál será el resultado?

```
console.log(0 && 1 || true);
```

false
 true
 0

91. ¿Cuál es una declaración válida de un array vacío?

```
let arr = ();
```

```
let arr = [];
```

```
let arr = {};
```

92. ¿Qué forma correcta declara un array con números?

```
let numeros = [1, 2, 3];
```

```
let numeros = (1, 2, 3);
```

```
let numeros = {1, 2, 3};
```

93. ¿Cuál es la forma válida de crear un array usando el constructor?

```
let arr = Array(1,2,3);
```

```
let arr = new Array(3);
```

```
let arr = Array[3];
```

94. ¿Qué declaración es correcta para un array de strings?

```
let frutas = ["manzana", "pera", "uva"];
```

```
let frutas = ("manzana", "pera", "uva");
```

```
let frutas = {"manzana", "pera", "uva"};
```

95. ¿Qué imprime este código?

```
let arr = [];
console.log(typeof arr);
```

- "list"
- "object"
- "array"

96. ¿Cuál es la forma correcta de declarar un array mixto?

```
let datos = {1, "dos", true};
```

```
let datos = [1, "dos", true];
```

```
let datos = (1, "dos", true);
```

97. ¿Qué resultado devuelve este código?

```
let arr = new Array("a", "b", "c");
console.log(arr.length);
```

- 3
 - 0
 - undefined
-

98. ¿Cuál de estas es la forma válida de declarar un array de un solo elemento?

```
let arr = {42};
```

```
let arr = (42);
```

```
let arr = [42];
```

99. ¿Qué valor imprime este código?

```
let arr = [10, 20, 30];
console.log(arr[1]);
```

- 30
 - 10
 - 20
-

100. ¿Cuál de estas declaraciones es **inválida**?

-

```
let arr = [1 2 3];
```

-

```
let arr = [1,2,3];
```

-

```
let arr = [];
```

101. ¿Qué valor imprime este código?

```
let arr = [10, 20, 30];
console.log(arr[0]);
```

- 10
 - 30
 - 20
-

102. ¿Qué valor imprime este código?

```
let arr = ["a", "b", "c"];
console.log(arr[2]);
```

- "b"
 - "c"
 - "a"
-

103. ¿Qué valor se obtiene?

```
let arr = [1, 2, 3];
console.log(arr[3]);
```

- 0
 - undefined
 - 3
-

104. ¿Qué imprime este código?

```
let arr = [5, 10, 15];
arr[1] = 50;
console.log(arr[1]);
```

- 15
 - 10
 - 50
-

105. ¿Cuál es el resultado?

```
let arr = [1, 2, 3];
arr[3] = 4;
console.log(arr.length);
```

- 3
 - 4
 - 5
-

106. ¿Qué imprime este código?

```
let arr = [7, 8, 9];
console.log(arr[arr.length - 1]);
```

- 9
 - 7
 - 8
-

107. ¿Qué resultado se obtiene?

```
let arr = [];
arr[2] = 100;
console.log(arr[1]);
```

- 100
 - undefined
 - 0
-

108. ¿Qué valor imprime?

```
let arr = ["x", "y", "z"];
console.log(arr.at(-1));
```

- "y"
 - "z"
 - "x"
-

109. ¿Qué imprime este código?

```
let arr = ["uno", "dos"];
arr[0] = "primero";
console.log(arr[0]);
```

- "dos"
- "primero"
- "uno"

110. ¿Qué resultado se obtiene?

```
let arr = [1, 2];
arr[5] = 42;
console.log(arr.length);
```

- 5
- 2
- 6

111. ¿Qué función agrega un elemento al final de un array?

- shift()
- pop()
- push()

112. ¿Qué función elimina el último elemento de un array?

- push()
- pop()
- unshift()

113. ¿Qué función agrega un elemento al inicio de un array?

- shift()
- push()
- unshift()

114. ¿Qué función elimina el primer elemento de un array?

- pop()
- shift()
- unshift()

115. ¿Qué función devuelve una copia de una parte del array sin modificar el original?

- `slice()`
 - `splice()`
 - `concat()`
-

116. ¿Qué función une dos arrays y devuelve uno nuevo?

- `join()`
 - `push()`
 - `concat()`
-

117. ¿Qué función transforma todos los elementos de un array aplicando una función?

- `filter()`
 - `map()`
 - `forEach()`
-

118. ¿Qué función devuelve un nuevo array con los elementos que cumplen una condición?

- `reduce()`
 - `filter()`
 - `map()`
-

119. ¿Qué función reduce un array a un único valor?

- `reduce()`
 - `map()`
 - `filter()`
-

120. ¿Qué función devuelve el primer elemento que cumple con una condición?

- `find()`
- `filter()`

`some()`

121. ¿Cuál es la forma correcta de declarar un objeto vacío?

```
let obj = [];
```

```
let obj = {};
```

```
let obj = ();
```

122. ¿Qué declaración es válida para un objeto con una propiedad `nombre`?

```
let persona = { nombre: "Juan" };
```

```
let persona = { "nombre", "Juan" };
```

```
let persona = (nombre: "Juan");
```

123. ¿Cuál es la forma correcta de declarar un objeto con varias propiedades?

```
let coche = (marca: "Toyota", año: 2020);
```

```
let coche = { marca = "Toyota", año = 2020 };
```

```
let coche = { marca: "Toyota", año: 2020 };
```

124. ¿Qué imprime este código?

```
let obj = {};
console.log(typeof obj);
```

- "array"
 - "object"
 - "null"
-

125. ¿Cuál es la forma válida de crear un objeto usando el constructor?

```
let obj = Object{};
```

```
let obj = Object[];
```

```
let obj = new Object();
```

126. ¿Cuál es la forma abreviada correcta cuando el nombre de la variable y la propiedad son iguales?

```
let nombre = "Ana";
let persona = { nombre };
console.log(persona.nombre);
```

- "nombre"
 - "Ana"
 - undefined
-

127. ¿Qué imprime este código?

```
let x = 10, y = 20;
let punto = { x, y };
console.log(punto);
```

-
- { x: "x", y: "y" }
- { x: 10, y: 20 }
- { "x": "y" }

128. ¿Qué declaración es equivalente?

```
let edad = 30;
let persona = { edad };
```

-
- let persona = { edad: "edad" };
-

```
let persona = { "edad" };
```

```
let persona = { edad: edad };
```

129. ¿Cuál es el valor de la propiedad?

```
let color = "rojo";
let coche = { color };
console.log(coche.color);
```

- undefined
- "rojo"
- "color"

130. ¿Qué imprime este código?

```
let a = 1, b = 2;
let numeros = { a, b, suma: a + b };
console.log(numeros.suma);
```

- 3
- "a+b"
- undefined

131. ¿Qué función devuelve un array con las claves de un objeto?

```
Object.entries()
```

```
Object.values()
```

`Object.keys()`

132. ¿Qué función devuelve un array con los valores de un objeto?

`Object.values()`

`Object.entries()`

`Object.keys()`

133. ¿Qué función devuelve un array de pares `[clave, valor]` ?

`Object.entries()`

`Object.keys()`

`Object.values()`

134. ¿Qué función evita que se agreguen nuevas propiedades a un objeto?

`Object.preventExtensions()`

`Object.seal()`

`Object.freeze()`

135. ¿Qué función congela un objeto para que no se pueda modificar?

`Object.assign()`

`Object.preventExtensions()`

`Object.freeze()`

136. ¿Qué función copia propiedades de un objeto a otro?

`Object.keys()`

`Object.values()`

`Object.assign()`

137. ¿Qué función crea un nuevo objeto con un prototipo específico?

`Object.defineProperty()`

`Object.create()`

`Object.assign()`

138. ¿Qué función define una nueva propiedad en un objeto?

`Object.defineProperty()`

`Object.freeze()`

`Object.create()`

139. ¿Qué función devuelve el prototipo de un objeto?

`Object.setPrototypeOf()`

`Object.getPrototypeOf()`

`Object.create()`

140. ¿Qué función sella un objeto, permitiendo modificar propiedades existentes pero no agregar ni eliminar?

`Object.seal()`

`Object.preventExtensions()`

`Object.freeze()`

141. ¿Qué valor imprime este código?

```
let [a, b] = [10, 20];
console.log(a);
```

 `undefined` `20` `10`

142. ¿Qué valor imprime este código?

```
let [x, y, z] = [1, 2, 3];
console.log(z);
```

- undefined
 - 3
 - 2
-

143. ¿Qué resultado imprime?

```
let [a, , c] = [1, 2, 3];
console.log(c);
```

- 1
 - 3
 - 2
-

144. ¿Qué valor obtiene `b`?

```
let [a, b = 5] = [10];
console.log(b);
```

- undefined
 - 5
 - 10
-

145. ¿Qué imprime este código?

```
let [x, ...rest] = [1, 2, 3, 4];
console.log(rest);
```

-

[3, 4]

[2, 3, 4]

[1, 2, 3]

146. ¿Qué valor imprime?

```
let [, , third] = ["a", "b", "c"];
console.log(third);
```

- "a"
 - "c"
 - "b"
-

147. ¿Qué resultado da este código?

```
let [x, y, z = 100] = [1, 2];
console.log(z);
```

- 100
 - undefined
 - 2
-

148. ¿Qué imprime?

```
let [a, b] = [1];
console.log(b);
```

- undefined
 - 0
 - null
-

149. ¿Qué resultado devuelve?

```
let [a, , b] = [10, 20, 30];
console.log(a, b);
```

- 20 30
 - 10 30
 - 10 20
-

150. ¿Qué valor imprime?

```
let [a, ...b] = [1];
console.log(b);
```

- []
 - [1]
 - undefined
-

151. ¿Qué valor imprime este código?

```
let { nombre } = { nombre: "Ana", edad: 25 };
console.log(nombre);
```

- undefined
 - 25
 - "Ana"
-

152. ¿Qué valor imprime?

```
let { edad } = { nombre: "Ana", edad: 25 };
console.log(edad);
```

- "Ana"
- 25

undefined

153. ¿Qué valor obtiene pais ?

```
let { pais = "Argentina" } = { nombre: "Ana" };
console.log(pais);
```

- "Argentina"
 - "Ana"
 - undefined
-

154. ¿Qué resultado devuelve este código?

```
let { nombre: n } = { nombre: "Luis" };
console.log(n);
```

- undefined
 - "Luis"
 - "nombre"
-

155. ¿Qué valor imprime?

```
let obj = { a: 1, b: 2, c: 3 };
let { b } = obj;
console.log(b);
```

- 1
 - 3
 - 2
-

156. ¿Qué resultado imprime?

```
let { x, y } = { x: 5 };
console.log(y);
```

undefined

0

5

157. ¿Qué imprime este código?

```
let { a = 10 } = { a: 0 };
console.log(a);
```

10

0

undefined

158. ¿Qué valor imprime?

```
let { nombre, edad } = { nombre: "Ana", edad: 30 };
console.log(edad);
```

undefined

"Ana"

30

159. ¿Qué imprime este código?

```
let { a, ...resto } = { a: 1, b: 2, c: 3 };
console.log(resto);
```

{ a: 1 }

{ b: 2, c: 3 }

{}

160. ¿Qué resultado se obtiene?

```
let { x, y = 50 } = { x: 10 };
console.log(x, y);
```

- 10 50
 - undefined 50
 - 10 undefined
-

161. ¿Cuál es la forma correcta de un `if` simple?

-

```
if (x > 10) { console.log("Mayor"); }
```

-

```
if x > 10 { console.log("Mayor"); }
```

-

```
if (x > 10) console.log("Mayor")
```

162. ¿Qué sintaxis es válida para un `if` con `else`?

-

```
if (x === 5) { console.log("Cinco"); } else { console.log("Otro"); }
```

-

```
elseif (x === 5) { console.log("Cinco"); } else { console.log("Otro"); }
```

-

```
if x === 5 console.log("Cinco"); else console.log("Otro")
```

163. ¿Cuál de estos `if` es incorrecto?

```
if (a == b) { console.log("Iguales"); }
```

```
if (a == b) console.log("Iguales");
```

```
if a == b console.log("Iguales");
```

164. ¿Qué imprime este código?

```
let n = 5;
if (n > 10) console.log("Mayor"); else console.log("Menor o igual");
```

-
- "Mayor"
 - undefined
 - "Menor o igual"

165. ¿Cuál es la forma correcta de un `if else if`?

```
if (x > 0) { console.log("Positivo"); }
else if (x < 0) { console.log("Negativo"); }
else { console.log("Cero"); }
```

```
if (x > 0) console.log("Positivo")
else (x < 0) console.log("Negativo")
else console.log("Cero")
```

- Ambas son válidas
-

166. ¿Qué imprime?

```
let edad = 18;  
if (edad >= 18) console.log("Adulto");
```

- "Adulto"
 undefined
 "18"
-

167. ¿Qué sintaxis es inválida?

```
if (condicion) { console.log("ok"); }
```

```
if condicion { console.log("ok"); }
```

```
if (condicion) console.log("ok");
```

168. ¿Qué resultado imprime?

```
let x = 0;  
if (x) console.log("Verdadero"); else console.log("Falso");
```

- "Falso"
 undefined
 "Verdadero"
-

169. ¿Qué imprime este código?

```
if ("texto") console.log("Verdadero"); else console.log("Falso");
```

- undefined
 - "Falso"
 - "Verdadero"
-

170. ¿Qué forma es válida para un `if` en una sola línea?

-

```
if x > 0 console.log("Positivo")
```

-

```
if (x > 0) console.log("Positivo");
```

-

```
if (x > 0) { console.log("Positivo") }
```

171. ¿Cuál es la forma correcta de un bucle `while`?

-

```
while (x < 10) {  
    console.log(x); x++;  
}
```

-

```
while (x < 10) {  
    console.log(x); x++;  
}
```

-

```
while x < 10 {  
    console.log(x); x++; }
```

```
while (x < 10)  
    console.log(x); x++;
```

172. ¿Qué imprime este código?

```
let i = 0;  
while (i < 3) {  
    console.log(i); i++;  
}
```

- 0 1 2
- 0 1 2 3
- 1 2 3

173. ¿Cuál de estas formas es incorrecta?

```
while (true) {  
    break;  
}
```

```
while (true) {  
    break;  
}
```

```
while (false) {  
    console.log("Nunca");  
}
```

```
while (true) {  
    break;  
}
```

174. ¿Qué valor imprime?

```
let x = 5;  
while (x > 0) {  
    x--;  
}  
console.log(x);
```

- 1
- 0
- 5

175. ¿Qué resultado obtiene este código?

```
let i = 0;  
while (i < 2) {  
    console.log(i++);  
}
```

- 0 1
- 1 2
- 0 1 2

176. ¿Cuál es la sintaxis válida?

```
while condicion { /* código */ }
```

```
while (condicion) { /* código */ }
```

 Ambas

177. ¿Qué imprime este bucle?

```
let i = 2;  
while (i) { console.log(i); i--; }
```

- 2 1
 - 2 1 0
 - 1 0
-

178. ¿Cuál es el resultado?

```
let i = 0;  
while (i < 0) { console.log(i); i++; }
```

- undefined
 - 0
 - Nada (no entra al bucle)
-

179. ¿Qué imprime este código?

```
let i = 3;  
while (i--) console.log(i);
```

- 3 2 1
 - 2 1 0
 - 3 2 1 0
-

180. ¿Qué forma es válida para un bucle `while` de una sola línea?

```
while (x < 5) x++;
```

```
while x < 5 { x++ }
```

```
while (x < 5) { x++ }
```

181. ¿Cuál es la forma básica de un bucle `for`?

```
for inicialización; condición; actualización { /* código */ }
```

```
for (inicialización; condición; actualización) { /* código */ }
```

```
for (inicialización condición actualización) { /* código */ }
```

182. ¿Qué imprime este código?

```
for (let i = 0; i < 3; i++) { console.log(i); }
```

 0 1 2 0 1 2 3 1 2 3

183. ¿Cuál de estas formas es válida para un bucle infinito?

```
for (;;) { }
```

```
for (true) { }
```

```
for () { }
```

184. ¿Qué imprime este código?

```
for (let i = 2; i > 0; i--) { console.log(i); }
```

 2 1 0 2 1 1 0

185. ¿Cuál es el resultado?

```
for (let i = 0; i < 2; ) { console.log(i); i++; }
```

 0 1 1 2 0 1 2

186. ¿Qué imprime este código?

```
for (let i = 0; i < 5; i += 2) { console.log(i); }
```

- 1 3 5
 - 0 1 2 3 4
 - 0 2 4
-

187. ¿Cuál es la sintaxis inválida?

-

```
for (let i = 0; i < 10) { }
```

-

```
for (let i = 0; i < 10; i++) { }
```

-

```
for (;;) { }
```

188. ¿Qué imprime este código?

```
for (let i = 3; i--; ) console.log(i);
```

- 2 1 0
 - 3 2 1 0
 - 3 2 1
-

189. ¿Qué valor tendrá `suma` ?

```
let suma = 0;
for (let i = 1; i <= 3; i++) {
    suma += i;
}
console.log(suma);
```

- 3

10

6

190. ¿Qué forma de `for` es válida en una sola línea?

```
for let i = 0; i < 3; i++ { console.log(i); }
```

```
for (let i = 0, i < 3, i++) { console.log(i); }
```

```
for (let i = 0; i < 3; i++) console.log(i);
```

191. ¿Qué hace el bucle `for...in` ?

- Itera sobre las propiedades enumerables de un objeto
 - Itera sobre los valores de un array
 - Itera sobre caracteres de un string
-

192. ¿Qué imprime este código?

```
let persona = { nombre: "Ana", edad: 25 };
for (let clave in persona) {
  console.log(clave);
}
```

Error

nombre edad

"Ana" 25

193. ¿Qué imprime este código?

```
let arr = ["a", "b", "c"];
for (let i in arr) {
  console.log(i);
}
```

- 0 1 2
 - Error
 - "a" "b" "c"
-

194. ¿Qué diferencia tiene `for...in` con `for...of`?

- `for...in` itera solo arrays, `for...of` solo objetos
 - No hay diferencia
 - `for...in` itera sobre índices o claves, `for...of` sobre valores
-

195. ¿Qué imprime este código?

```
let obj = { a: 1, b: 2 };
for (let clave in obj) {
  console.log(obj[clave]);
}
```

- 1 2
 - "a" "b"
 - undefined
-

196. ¿Qué hace el bucle `for...of`?

- Itera sobre los valores de un objeto iterable (arrays, strings, etc.)
 - Itera sobre las propiedades enumerables de un objeto
 - Itera sobre índices únicamente
-

197. ¿Qué imprime este código?

```
let arr = [10, 20, 30];
for (let valor of arr) {
  console.log(valor);
}
```

- Error
 - 0 1 2
 - 10 20 30
-

198. ¿Qué imprime este código?

```
let texto = "JS";
for (let char of texto) {
  console.log(char);
}
```

- "JS"
 - 0 1
 - J S
-

199. ¿Qué diferencia tiene `for...of` con `for...in`?

- No hay diferencia
 - `for...of` itera sobre valores, `for...in` sobre índices o claves
 - `for...of` solo funciona con objetos, `for...in` con arrays
-

200. ¿Qué imprime este código?

```
let arr = ["a", "b"];
for (let valor of arr) {
  console.log(valor.toUpperCase());
}
```

- undefined
- "a" "b"
- A B

201. ¿Qué hace la instrucción `break` dentro de un bucle?

- Termina inmediatamente el bucle
 - Reinicia el bucle desde el principio
 - Salta la iteración actual y continúa con la siguiente
-

202. ¿Qué hace la instrucción `continue` dentro de un bucle?

- Salta la iteración actual y continúa con la siguiente
 - Termina inmediatamente el bucle
 - Sale del programa
-

203. ¿Qué imprime este código?

```
for (let i = 0; i < 5; i++) {  
  if (i === 3) break;  
  console.log(i);  
}
```

- 0 1 2 3 4
 - 0 1 2
 - 0 1 2 3
-

204. ¿Qué imprime este código?

```
for (let i = 0; i < 5; i++) {  
  if (i === 2) continue;  
  console.log(i);  
}
```

- 0 1 2 3 4
 - 0 1 3 4
 - 1 2 3 4
-

205. ¿Qué imprime este bucle?

```
let i = 0;
while (i < 5) {
    i++;
    if (i === 4) break;
    console.log(i);
}
```

- 1 2 3
 - 1 2 3 4
 - 0 1 2 3
-

206. ¿Cuál es la forma básica correcta de un `switch`?

-

```
switch expresion {
    case valor1: /* código */ break;
}
```

-

```
switch (expresion) {
    case valor1: /* código */ break;
    case valor2: /* código */ break;
    default: /* código */
}
```

-

```
switch (expresion)
    case valor1: /* código */
```

207. ¿Qué imprime este código?

```
let x = 2;
switch (x) {
  case 1: console.log("Uno"); break;
  case 2: console.log("Dos"); break;
  default: console.log("Otro");
}
```

- "Otro"
 - "Uno"
 - "Dos"
-

?208. ¿Qué sucede si falta el `break` en un caso?

- Termina el `switch` automáticamente
 - Ejecuta el caso siguiente también
 - Lanza un error
-

?209. ¿Qué imprime este código?

```
let color = "rojo";
switch (color) {
  case "azul": console.log("Azul"); break;
  case "rojo": console.log("Rojo"); break;
  default: console.log("Otro");
}
```

- "Rojo"
 - "Otro"
 - "Azul"
-

?210. ¿Qué imprime este código?

```
let n = 10;
switch (n) {
  case 5: console.log("Cinco"); break;
  default: console.log("Otro");
}
```

- "Otro"
 - undefined
 - "Cinco"
-

?11. ¿Cuál es la forma correcta de declarar una función?

```
function sumar(a, b) {  
    return a + b;  
}
```

```
function = sumar(a, b) {  
    return a + b;  
}
```

```
func sumar(a, b) {  
    return a + b;  
}
```

?12. ¿Qué imprime este código?

```
function saludo() { return "Hola"; }  
console.log(saludo());
```

- undefined
 - "Hola"
 - saludo
-

?13. ¿Qué ocurre si no se usa `return` dentro de una función?

- Devuelve undefined

- Devuelve `null`
 - Lanza un error
-

?14. ¿Qué forma es válida para una función sin parámetros?

```
function() mostrar {  
    console.log("Ok");  
}
```

```
function mostrar {  
    console.log("Ok");  
}
```

```
function mostrar() {  
    console.log("Ok");  
}
```

?15. ¿Qué imprime este código?

```
function f() {}  
console.log(f());
```

- `null`
 - `0`
 - `undefined`
-

?16. ¿Qué imprime este código?

```
function sumar(a, b) {  
    return a + b;  
}  
console.log(sumar(2, 3));
```

- 5
 - undefined
 - 23
-

?17. ¿Qué valor devuelve?

```
function f(x = 10) {  
    return x;  
}  
console.log(f());
```

- undefined
 - null
 - 10
-

?18. ¿Qué imprime este código?

```
function f(a, b = 2) {  
    return a * b;  
}  
console.log(f(5));
```

- 5
 - 10
 - undefined
-

?19. ¿Qué ocurre si se pasan más parámetros de los declarados?

```
function f(a, b) {  
    return a + b;  
}  
console.log(f(1, 2, 3));
```

- 3
 - 6
 - 1
-

?20. ¿Qué imprime este código?

```
function f(a, b) { console.log(arguments.length); }  
f(1, 2, 3, 4);
```

- 2
 - undefined
 - 4
-

?21. ¿Qué valor imprime?

```
function f(...args) { return args.length; }  
console.log(f(1, 2, 3));
```

- undefined
 - 3
 - 2
-

?22. ¿Qué imprime?

```
function f(a, ...rest) { return rest; }  
console.log(f(1, 2, 3));
```

- [3]
- [1, 2, 3]
- [2, 3]

?23. ¿Qué ocurre si no se pasan parámetros?

```
function f(a, b) { return a + b; }
console.log(f());
```

- `Nan`
 - `0`
 - `undefined`
-

?24. ¿Qué imprime este código?

```
function f(a = 1, b = 2) { return a + b; }
console.log(f(undefined, 5));
```

- `7`
 - `6`
 - `3`
-

?25. ¿Qué es una función anidada?

- Una función que se ejecuta varias veces
 - Una función definida dentro de otra función
 - Una función sin nombre
-

?26. ¿Qué imprime este código?

```
function externa() {
  function interna() {
    return "Hola";
  }
  return interna();
}
console.log(externa());
```

- `undefined`

- Error
 - "Hola"
-

?27. ¿Qué valor imprime?

```
function externa() {  
    let x = 10;  
    function interna() {  
        return x;  
    }  
    return interna();  
}  
console.log(externa());
```

- Error
 - undefined
 - 10
-

?28. ¿Qué ocurre si intentamos llamar a la función interna desde fuera?

```
function externa() {  
    function interna() {  
        return 5;  
    }  
}  
console.log(interna());
```

- 5
 - Error (interna no está definida en el ámbito global)
 - undefined
-

?29. ¿Qué imprime este código?

```
function externa() {  
    function interna(y) {  
        return y * 2;  
    }  
    return interna(4);  
}  
console.log(externa());
```

- 4
 - 8
 - undefined
-

?30. ¿Qué es la recursividad en programación?

- Cuando una función se llama a sí misma
 - Cuando una función se ejecuta varias veces en un bucle
 - Cuando una función contiene otra función
-

?31. ¿Qué imprime este código?

```
function cuentaAtras(n) {  
    if (n === 0) return "Fin";  
    return n + " " + cuentaAtras(n - 1);  
}  
console.log(cuentaAtras(3));
```

- "3 2 1 Fin"
 - "Fin"
 - "3 2 1 0"
-

?32. ¿Cuál es el caso base en una función recursiva?

- La parte que llama de nuevo a la función
 - La condición que detiene la recursión
 - El primer parámetro de la función
-

233. ¿Qué imprime este código?

```
function factorial(n) {  
    if (n === 1) return 1;  
    return n * factorial(n - 1);  
}  
console.log(factorial(4));
```

- 24
 - 10
 - 4
-

234. ¿Qué ocurre si a una función recursiva le falta el caso base?

- Devuelve `undefined`
 - Se detiene automáticamente después de 10 llamadas
 - Se produce un bucle infinito y un error de pila (stack overflow)
-

235. ¿Qué imprime este código?

```
function f() {  
    let x = 10; console.log(x);  
}  
f();
```

- 10
 - Error
 - `undefined`
-

236. ¿Qué ocurre?

```
function f() {  
    let x = 5;  
}  
console.log(x);
```

- Error (x no está definida fuera de la función)

undefined

5

?37. ¿Qué valor imprime?

```
let x = 1;
function f() {
    let x = 2; console.log(x);
}
f();
```

undefined

1

2

?38. ¿Qué imprime este código?

```
let x = 1;
function f() {
    console.log(x);
}
f();
```

Error

1

undefined

?39. ¿Qué ocurre?

```
function f() {
    y = 10;
}
f();
console.log(y);
```

10

undefined

Error

?40. ¿Qué imprime?

```
function f() {  
    var x = 7;  
}  
console.log(typeof x);
```

- "undefined"
 - "number"
 - "object"
-

?41. ¿Qué valor imprime este código?

```
function mostrar({ nombre }) {  
    console.log(nombre);  
}  
mostrar({ nombre: "Ana", edad: 25 });
```

- undefined
 - "Ana"
 - 25
-

?42. ¿Qué valor imprime?

```
function f({ x, y }) {  
    console.log(x + y);  
}  
f({ x: 2, y: 3 });
```

- undefined
 - 5
 - 23
-

243. ¿Qué resultado da este código?

```
function f({ a = 10, b = 20 }) {  
    console.log(a, b);  
}  
f({ a: 5 });
```

- 5 undefined
 - 5 20
 - 10 20
-

244. ¿Qué imprime este código?

```
function componente({ titulo }) {  
    console.log(titulo);  
}  
componente({ titulo: "React" });
```

- undefined
 - "titulo"
 - "React"
-

245. ¿Qué valor imprime?

```
function f([a, b]) {  
    console.log(a, b);  
}  
f([1, 2]);
```

- 2 1
 - 1 2
 - undefined undefined
-

246. ¿Qué resultado imprime?

```
function f([primero, , tercero]) {
    console.log(tercero);
}
f([10, 20, 30]);
```

- 10
 - 30
 - 20
-

247. ¿Qué imprime este código?

```
function f({ nombre = "Invitado" }) {
    console.log(nombre);
}
f({});
```

- undefined
 - "Invitado"
 - "nombre"
-

248. ¿Qué valor imprime?

```
function f({ a, ...resto }) { console.log(resto); }
f({ a: 1, b: 2, c: 3 });
```

- { b: 2, c: 3 }
 - {}
 - { a: 1 }
-

249. ¿Qué imprime este código?

```
function f([x, ...resto]) {
    console.log(resto);
}
f([1, 2, 3]);
```

- [3]
 - [1, 2, 3]
 - [2, 3]
-

?250. ¿Qué imprime este ejemplo típico de React?

```
function Boton({ texto }) {  
  console.log(texto);  
}  
Boton({ texto: "Click aquí" });
```

- "Click aquí"
 - undefined
 - "texto"
-

?251. ¿Cuál es la forma correcta de declarar una función anónima y asignarla a una variable?

-

```
let f = function() {  
  return "Hola";  
};
```

-

```
function = () {  
  return "Hola";  
};
```

-

```
let f = function; {  
  return "Hola";  
}
```

?252. ¿Qué imprime este código?

```
let f = function(x) {  
    return x * 2;  
};  
console.log(f(5));
```

- 10
 - 5
 - undefined
-

253. ¿Cuál es una forma válida de usar una función anónima como parámetro?

-

```
setTimeout(() => { console.log("Hola"); }, 1000);
```

-

```
setTimeout(function() { console.log("Hola"); }, 1000);
```

-

```
setTimeout(function; { console.log("Hola"); }, 1000);
```

254. ¿Qué imprime este código?

```
[1, 2, 3].forEach(function(n) { console.log(n); });
```

- Error
 - 0 1 2
 - 1 2 3
-

255. ¿Qué valor devuelve?

```
let resultado = [1, 2, 3].map(function(n) { return n * 2; });  
console.log(resultado);
```

[2, 4, 6]

[1, 2, 3]

[n * 2]

?256. ¿Cuál es la forma correcta de pasar una función anónima a `filter`?

`let pares = [1, 2, 3, 4].filter(n) { return n % 2 === 0; };`

`let pares = [1, 2, 3, 4].filter(function(n) { return n % 2 === 0; });`

`let pares = [1, 2, 3, 4].filter(function => n % 2 === 0);`

?257. ¿Qué imprime este código?

```
let f = function(nombre) { return "Hola " + nombre; };
console.log(f("Ana"));
```

- "Hola Ana"
- "Ana"
- undefined

?258. ¿Qué forma es válida para ejecutar una función anónima inmediatamente?

```
(function() { console.log("Ejecutada"); })();
```

```
((() console.log("Ejecutada")); )();
```

```
function() { console.log("Ejecutada"); }();
```

?261. ¿Cuál es la forma correcta de una función flecha que suma dos números?

```
let sumar = (a, b) -> a + b;
```

```
let sumar = (a, b) { return a + b; };
```

```
let sumar = (a, b) => a + b;
```

?262. ¿Qué imprime este código?

```
let f = () => "Hola";
console.log(f());
```

 f "Hola" undefined

?63. ¿Qué valor devuelve?

```
let cuadrado = x => x * x;  
console.log(cuadrado(4));
```

- 16
- 8
- 4

?64. ¿Qué imprime este código?

```
let saludo = nombre => "Hola " + nombre;  
console.log(saludo("Ana"));
```

- "Hola Ana"
- undefined
- "Ana"

?65. ¿Cuál es la forma correcta de una función flecha sin parámetros?

-

```
let f = {} => 42;
```

-

```
let f = () => 42;
```

-

```
let f = => 42;
```

?66. ¿Qué imprime este código?

```
let suma = (a, b) => { return a + b; };
console.log(suma(2, 3));
```

- 5
 - undefined
 - 23
-

?267. ¿Qué imprime este código?

```
let doble = n => n * 2;
console.log(doble(7));
```

- 7
 - 14
 - undefined
-

?268. ¿Qué valor imprime?

```
let resta = (a, b) => a - b;
console.log(resta(10, 3));
```

- 13
 - undefined
 - 7
-

?269. ¿Qué resultado devuelve?

```
let obj = {
  metodo: () => this.valor
};
console.log(obj.metodo());
```

- 10
- valor
- undefined

270. ¿Qué ventaja tiene usar funciones flecha?

- Tienen acceso a la API de React
 - Son más rápidas que las funciones normales
 - No tienen su propio `this`, lo que puede evitar confusiones con el contexto
-

271. ¿Qué imprime este código?

```
let arr = [1, 2, 3];
let resultado = arr.map(x => x * 2);
console.log(resultado);
```

[x * 2]

[1, 2, 3]

[2, 4, 6]

272. ¿Qué resultado devuelve?

```
let arr = [1, 2, 3, 4];
let pares = arr.filter(x => x % 2 === 0);
console.log(pares);
```

[1, 3]

[0, 2, 4]

[2, 4]

?73. ¿Qué imprime este código?

```
let arr = [1, 2, 3];
let suma = arr.reduce((a, b) => a + b, 0);
console.log(suma);
```

- 6
 - 123
 - 0
-

?74. ¿Qué valor devuelve?

```
let arr = ["a", "b", "c"];
let joined = arr.join("-");
console.log(joined);
```

- ["a","b","c"]
 - "abc"
 - "a-b-c"
-

?75. ¿Qué imprime este código?

```
let arr = [10, 20, 30];
let encontrado = arr.find(x => x > 15);
console.log(encontrado);
```

- 20
- 30
- 10

276. ¿Qué resultado devuelve?

```
let arr = [1, 2, 3, 4];
let existe = arr.some(x => x > 3);
console.log(existe);
```

- true
- false
- undefined

277. ¿Qué imprime este código?

```
let arr = [1, 2, 3, 4];
let todos = arr.every(x => x < 5);
console.log(todos);
```

- undefined
- false
- true

278. ¿Qué valor imprime?

```
let arr = [5, 10, 15];
let total = arr.reduce((acc, val) => acc + val);
console.log(total);
```

- 25
- [5, 10]
- 30

279. ¿Qué imprime este código?

```
let arr = [1, 2, 3];
arr.forEach(x => console.log(x * 2));
```

- 1 2 3
 - [2, 4, 6]
 - 2 4 6
-

?80. ¿Qué resultado devuelve?

```
let arr = [1, 2, 3, 4, 5];
let recortado = arr.slice(1, 4);
console.log(recortado);
```

-

[2, 3, 4]

-

[1, 2, 3, 4]

-

[3, 4, 5]

?81. ¿Cuál es la forma correcta de declarar una clase?

-

```
function class Persona {}
```

-

```
class Persona {}
```

-

```
class = Persona {}
```

282. ¿Cómo se define un constructor en una clase?

```
class Persona {  
    construct(nombre) { this.nombre = nombre; }  
}
```

```
class Persona {  
    function constructor(nombre) { this.nombre = nombre; }  
}
```

```
class Persona {  
    constructor(nombre) { this.nombre = nombre; }  
}
```

283. ¿Qué imprime este código?

```
class Animal {  
    constructor(tipo) { this.tipo = tipo; }  
}  
let perro = new Animal("perro");  
console.log(perro.tipo);
```

- "perro"
 - undefined
 - "Animal"
-

284. ¿Cómo se declara un método en una clase?

```
class Persona {  
    function hablar() { return "Hola"; }  
}
```

```
class Persona {  
    hablar() { return "Hola"; }  
}
```

```
class Persona {  
    hablar: function() { return "Hola"; }  
}
```

285. ¿Qué caracteriza a la programación **reactiva**?

- Los datos nunca cambian en el tiempo
 - La interfaz responde automáticamente a los cambios en el estado o datos
 - El programador debe refrescar manualmente la pantalla
-

286. ¿Qué ventaja tiene el enfoque declarativo en React?

- Permite manipular directamente el DOM con `document.getElementById`
 - Requiere más líneas de código que el enfoque imperativo
 - Hace el código más predecible y fácil de entender
-

287. ¿Qué ocurre cuando cambia el estado en un componente de React?

- El programador debe llamar manualmente a `render()`
 - No ocurre nada hasta reiniciar la aplicación
 - La interfaz se actualiza automáticamente (reactividad)
-

288. ¿Cuál de las siguientes es una característica de la **programación reactiva** en React?

- Los componentes se vuelven a renderizar automáticamente al cambiar el estado o props
 - Solo se renderiza una vez y luego nunca cambia
 - Se deben usar funciones externas para refrescar la interfaz
-

421. ¿Qué es un componente en React?

- Un archivo CSS para estilos
 - Una base de datos para guardar información
 - Una función o clase que devuelve elementos de interfaz de usuario
-

422. ¿Cómo se declara un **componente funcional** básico en React?

```
function Hola {  
  <h1>Hola</h1>  
}
```

```
function Hola() {  
  return <h1>Hola</h1>;  
}
```

```
let Hola = <h1>Hola</h1>;
```

423. ¿Cómo se usan los componentes en React dentro de JSX?

```
<Componente />
```

{Componente}

Componente();

424. ¿Cuál es la forma correcta de declarar un componente como **arrow function**?

const Hola => <h1>Hola</h1>;

const Hola = () { return <h1>Hola</h1>; };

const Hola = () => <h1>Hola</h1>;

425. ¿Qué distingue a un **componente de clase**?

- Usa la palabra clave `class` y extiende `React.Component`
- Solo puede devolver cadenas de texto
- No puede tener estado

426. ¿Cuál es la sintaxis básica de un **componente de clase**?

function class Hola() {
 return <h1>Hola</h1>;
}

```
class Hola extends React.Component {  
  render() {  
    return <h1>Hola</h1>;  
  }  
}
```

```
class Hola {  
  return <h1>Hola</h1>;  
}
```

427. ¿Qué característica tienen los **componentes reutilizables**?

- No aceptan parámetros
 - Solo se pueden declarar una vez
 - Pueden usarse varias veces en diferentes partes de la aplicación
-

428. ¿Cómo se pueden pasar datos a un componente?

- Usando **props**
 - Modificando el DOM manualmente
 - Usando directamente variables globales
-

429. ¿Qué imprime este código?

```
function Saludo(props) {  
  return <h1>Hola {props.nombre}</h1>;  
}  
<Saludo nombre="Ana" />
```

- Hola Ana
 - Hola
 - Hola {props.nombre}
-

430. ¿Cuál es la ventaja principal de usar componentes en React?

- Permiten usar Java en lugar de JavaScript
 - Reutilización y modularidad del código
 - Se necesita menos HTML
-

431. ¿Cómo se pasan parámetros a un componente en React?

- Modificando directamente el DOM
 - Usando variables globales
 - A través de **props**
-

432. ¿Qué imprime este código?

```
function Saludo(props) {  
  return <h1>Hola {props.nombre}</h1>;  
}  
<Saludo nombre="Ana" />
```

- Hola
 - Hola Ana
 - Hola {props.nombre}
-

433. ¿Cómo se define un evento `onClick` en React?

-

```
<button onClick="miFuncion()">Click</button>
```

-

```
<button onClick={miFuncion}>Click</button>
```

-

```
<button onclick="miFuncion()">Click</button>
```

434. ¿Qué imprime este código?

```
function Boton() {  
  function manejarClick() {  
    console.log("Clickeado");  
  }  
  return <button onClick={manejarClick}>Click</button>;  
}
```

- "Clickeado" inmediatamente
- Error
- "Clickeado" al hacer click

435. ¿Cómo se pasa un parámetro a un evento en React?

-

```
<button onclick="miFuncion('Hola')">Click</button>
```

-

```
<button onClick={() => miFuncion("Hola")}>Click</button>
```

-

```
<button onClick={miFuncion("Hola")}>Click</button>
```

436. ¿Qué imprime este código?

```
function Mostrar(props) {  
  return <p>{props.mensaje}</p>;  
}  
<Mostrar mensaje="Hola React" />
```

- Hola React
- mensaje

undefined

437. ¿Qué ocurre si se pasa una función como prop?

- Se convierte en string automáticamente
 - Se ejecuta dentro del componente hijo
 - Lanza un error
-

438. ¿Qué imprime este código?

```
function Hijo(props) {  
  return <button onClick={props.onSaludar}>Saludar</button>;  
}  
function Padre() {  
  function saludar() { console.log("Hola desde padre"); }  
  return <Hijo onSaludar={saludar} />;  
}
```

- Nada ocurre
 - "Hola desde padre" al hacer click
 - Error
-

439. ¿Cómo se pueden pasar múltiples props a un componente?

```
<Componente {nombre: "Ana", edad: 25} />
```

```
<Componente (nombre="Ana", edad=25) />
```

```
<Componente nombre="Ana" edad={25} />
```

140. ¿Cuál es la forma correcta de manejar eventos en React?

- Usar HTML puro sin eventos
 - Usar camelCase (`onClick`, `onChange`) y pasar funciones
 - Usar atributos en minúsculas (`onclick`) y pasar strings
-

141. ¿Qué son los Hooks en React?

- Una nueva forma de declarar clases en React
 - Funciones especiales que permiten usar estado y otras características en componentes funcionales
 - Un sistema de estilos en React
-

142. ¿En qué versión de React se introdujeron los Hooks?

- React 15.0
 - React 16.8
 - React 18
-

143. ¿Cuál es la regla principal al usar Hooks?

- Solo se pueden usar en el nivel superior de un componente
 - Pueden declararse dentro de cualquier función
 - Pueden usarse dentro de cualquier bucle o condicional
-

144. ¿Cuál es un Hook básico para manejar el estado?

- `useEffect`
 - `useContext`
 - `useState`
-

145. ¿Qué Hook se usa para manejar efectos secundarios (ejemplo: llamadas a APIs)?

- `useReducer`

- `useEffect`
 - `useState`
-

146. ¿Qué permite hacer el Hook `useContext` ?

- Crear un nuevo estado local
 - Definir un ciclo de vida
 - Acceder a un contexto sin necesidad de usar un Consumer
-

147. ¿Cuál es la ventaja principal de los Hooks?

- Reemplazan totalmente a los componentes de clase
 - Permiten usar estado y lógica reutilizable en componentes funcionales
 - Reducen el tamaño de los archivos CSS
-

148. ¿Qué ocurre si se usan Hooks dentro de un condicional?

- Puede romper las reglas de los Hooks y causar errores
 - Se convierten en funciones normales
 - Funciona sin problemas siempre
-

151. ¿Qué es `useState` ?

- Un Hook que permite añadir estado a los componentes funcionales
 - Una función para renderizar componentes
 - Un método exclusivo de clases
-

152. ¿Cómo se importa `useState` ?

-

```
import { useState } from "react";
```

-

```
const useState = require("react");
```

```
import useState from "react";
```

453. ¿Qué devuelve la llamada a `useState`?

- Un array con el valor actual y una función para actualizarlo
- Solo el valor inicial
- Un objeto con el estado y un método `setState`

454. ¿Qué imprime este código?

```
const [contador, setContador] = useState(0);
console.log(contador);
```

- `undefined`
- Error
- `0`

455. ¿Cómo se actualiza el estado con `useState`?

- Usando la función de actualización (`setAlgo`)
- Modificando directamente la variable
- Reiniciando el componente

456. ¿Qué imprime este código después de un click?

```
const [contador, setContador] = useState(0);
<button onClick={() => setContador(contador + 1)}>
  Incrementar
</button>
```

- El valor de `contador` aumenta en 1
 - Da error
 - Siempre imprime 0
-

457. ¿Qué ocurre si se llama a `setEstado` varias veces seguidas?

- Ejecuta cada una inmediatamente en orden
 - Lanza un error
 - React agrupa las actualizaciones para optimizar
-

458. ¿Cuál es la sintaxis correcta para usar un valor inicial dinámico en `useState` ?

```
const [valor, setValor] = useState(() => calcularInicial());
```

```
const [valor, setValor] = useState = calcularInicial();
```

```
const [valor, setValor] = useState{calcularInicial()};
```

459. ¿Qué pasa si se actualiza el estado con el mismo valor?

- React no vuelve a renderizar el componente
 - Siempre vuelve a renderizar
 - Lanza un error
-

460. ¿Qué imprime este código?

```
const [texto, setTexto] = useState("Hola");
setTexto("Mundo");
console.log(texto);
```

- Error
 - "Mundo" inmediatamente
 - "Hola" en el render actual
-

461. ¿Cómo se actualiza correctamente un objeto en `useState` ?

-

```
setEstado({ ...estado, nuevoValor: 123 });
```

-

```
setEstado(estado.nuevoValor = 123);
```

-

```
estado.nuevoValor = 123;
```

462. ¿Qué ocurre cuando se modifica directamente un objeto en `useState` ?

- No se re-renderiza el componente
 - React actualiza automáticamente
 - Se lanza un error
-

463. ¿Cuál es la forma correcta de agregar un elemento a un array en el estado?

-

```
setLista(lista.push(nuevoElemento));
```

-

```
lista.push(nuevoElemento);
```

```
setLista([...lista, nuevoElemento]);
```

464. ¿Qué imprime este código?

```
const [usuario, setUsuario] = useState({ nombre: "Ana", edad: 20 });
setUsuario({ ...usuario, edad: 21 });
console.log(usuario.edad);
```

- Error
- 20
- 21

465. ¿Cómo se elimina un elemento de un array en el estado?

```
setLista(lista.filter(item => item !== valor));
```

```
delete lista[valor];
```

```
lista.remove(valor);
```

466. ¿Cómo se actualiza una propiedad anidada en un objeto con `useState` ?

```
setEstado(estado.direccion.ciudad = "Roma");
```

```
setEstado({ ...estado, direccion: { ...estado.direccion, ciudad: "Roma" } });
```

```
estado.direccion.ciudad = "Roma";
```

467. ¿Qué ocurre si se reemplaza el array en `useState` sin usar spread?

- Lanza un error
 - Se conservan los elementos previos automáticamente
 - Se pierde el contenido anterior
-

468. ¿Cómo se agrega un objeto nuevo a un array en el estado?

```
usuarios.push({ id: 3, nombre: "Luis" });
```

```
setUsuarios(usuarios.add({ id: 3, nombre: "Luis" }));
```

```
setUsuarios([...usuarios, { id: 3, nombre: "Luis" }]);
```

469. ¿Qué pasa si se llama a `setEstado` con el mismo objeto sin crear una copia?

- React siempre vuelve a renderizar
- Da un error
- React no detecta cambios y no renderiza de nuevo

470. ¿Qué imprime este código?

```
const [lista, setLista] = useState([1, 2, 3]);
setLista([...lista, 3]);
console.log(lista);
```

- [1, 2, 3, 3]
 - Error
 - [1, 2, 3]
-

471. ¿Qué es `useEffect` en React?

- Un método exclusivo de clases
 - Una función para actualizar el estado
 - Un Hook que maneja efectos secundarios en componentes funcionales
-

472. ¿Cómo se importa `useEffect`?

-

```
const useEffect = require("react");
```

-

```
import useEffect from "react";
```

-

```
import { useEffect } from "react";
```

473. ¿Qué imprime este código?

```
useEffect(() => {
  console.log("Efecto ejecutado");
});
```

- Nunca imprime nada
 - "Efecto ejecutado" solo en el primer render
 - "Efecto ejecutado" en cada render
-

474. ¿Cómo se ejecuta un efecto solo una vez al montar el componente?

- Usando `useEffectOnce`
 - Usando `useEffect(() => {...})` sin dependencias
 - Pasando un array vacío como dependencia: `useEffect(() => {...}, [])`
-

475. ¿Qué ocurre si se pasa un array con dependencias a `useEffect` ?

- El efecto se ejecuta cuando cambian esas dependencias
 - El efecto nunca se ejecuta
 - El efecto se ejecuta siempre en cada render
-

476. ¿Qué imprime este código?

```
const [contador, setContador] = useState(0);
useEffect(() => {
  console.log("Cambio en contador");
}, [contador]);
```

- "Cambio en contador" cada vez que cambia `contador`
 - "Cambio en contador" solo una vez
 - Nunca imprime nada
-

477. ¿Cómo se limpia un efecto en `useEffect` ?

- Retornando una función de limpieza dentro del callback
- Llamando a `clearEffect()`

- Usando `useCleanup`
-

478. ¿Qué ocurre con este código?

```
useEffect(() => {
  const id = setInterval(() => console.log("Tick"), 1000);
  return () => clearInterval(id);
}, []);
```

- Imprime "Tick" una vez y se detiene
 - Da un error
 - Imprime "Tick" cada segundo y se limpia al desmontar el componente
-

479. ¿Qué problema puede ocurrir si no se limpian los efectos correctamente?

- Fugas de memoria o comportamientos inesperados
 - El estado deja de actualizarse
 - El componente no se renderiza
-

480. ¿Qué imprime este código?

```
const [x, setX] = useState(0);
useEffect(() => {
  console.log("Render con x:", x);
}, [x]);
setX(5);
```

- "Render con x: 0" y luego "Render con x: 5"
 - Solo "Render con x: 0"
 - Error
-

481. ¿Qué es `useContext` en React?

- Un Hook que permite acceder a un contexto sin necesidad de usar un Consumer
- Una función para manejar efectos secundarios

- Un método exclusivo de clases
-

482. ¿Cómo se importa `useContext` ?

-

```
import useContext from "react";
```

-

```
const useContext = require("react");
```

-

```
import { useContext } from "react";
```

483. ¿Qué se necesita antes de usar `useContext` ?

- Importar un reducer
 - Crear un contexto con `React.createContext()`
 - Definir un `useState`
-

484. ¿Cómo se accede a un valor de contexto con `useContext` ?

-

```
const valor = getContexto(MiContexto);
```

-

```
const valor = useContext(MiContexto);
```

-

```
const valor = MiContexto();
```

485. ¿Qué imprime este código?

```
const TemaContext = React.createContext("claro");
function Componente() {
  const tema = useContext(TemaContext);
  return <p>{tema}</p>;
}
```

- "claro"
- undefined
- Error

486. ¿Qué ocurre si un componente no está envuelto en un `Provider` ?

- Obtiene `null`
- Obtiene el valor por defecto del contexto
- Siempre da error

487. ¿Cómo se define un `Provider` de contexto?

-

```
<MiContexto value="algo">
  <App />
</MiContexto>
```

-

```
<Provider contexto={MiContexto} valor="algo">
  <App />
</Provider>
```

-

```
<MiContexto.Provider value="algo">
  <App />
</MiContexto.Provider>
```

488. ¿Qué imprime este código si la URL es /user/42 ?

```
function User() {  
  const params = useParams();  
  return <p>{params.id}</p>;  
}
```

- 42
- undefined
- user

489. ¿Qué ocurre si se cambia el valor del contexto en el Provider ?

- Lanza un error
- Nada cambia hasta reiniciar la aplicación
- Los componentes que usan useContext se re-renderizan con el nuevo valor

491. ¿Qué es useRef en React?

- Un método exclusivo de clases
- Un Hook que devuelve un objeto mutable con la propiedad .current
- Una función para crear estado

492. ¿Cómo se importa useRef ?

-

```
import { useRef } from "react";
```

-

```
const useRef = require("react");
```

-

```
import useRef from "react";
```

493. ¿Qué valor inicial tiene una referencia creada con `useRef()` ?

- `0`
- `null`
- `undefined`

494. ¿Qué imprime este código?

```
const ref = useRef(5);
console.log(ref.current);
```

- `5`
- `undefined`
- Error

495. ¿Qué ocurre cuando se actualiza `.current` de un `useRef` ?

- React actualiza automáticamente
- Se lanza un error
- No causa re-render del componente

496. ¿Cuál es un uso común de `useRef` ?

- Referenciar elementos del DOM
- Crear un estado global
- Definir reducers

497. ¿Qué imprime este código?

```
function Componente() {  
  const inputRef = useRef(null);  
  return <input ref={inputRef} />;  
}
```

- Error
 - Una referencia al elemento `<input>`
 - `null`
-

498. ¿Qué ocurre si se cambia manualmente `ref.current`?

- Lanza un error
 - Se vuelve a renderizar automáticamente
 - Se actualiza el valor pero no se renderiza el componente
-

499. ¿Cómo se usa `useRef` para almacenar un valor que persiste entre renders sin causar renderizados?

-

```
const contador = useRefState(0);
```

-

```
const contador = useRef(0);  
contador.current += 1;
```

-

```
const [contador, setContador] = useRef(0);
```

500. ¿Qué imprime este código?

```
const ref = useRef("Hola");
ref.current = "Mundo";
console.log(ref.current);
```

- undefined
 - "Hola"
 - "Mundo"
-

501. ¿Qué es Vite?

- Una herramienta de construcción rápida para proyectos frontend
 - Un framework de backend
 - Una base de datos
-

502. ¿Cómo se crea un nuevo proyecto con Vite usando npm?

-

```
npm create vite@latest
```

-

```
npx start next
```

-

```
npm init react-app
```

503. ¿Qué comando se usa para entrar en la carpeta del proyecto recién creado?

- cd nombre-proyecto
-

```
npm cd nombre-proyecto
```

- cd vite

504. ¿Qué comando se usa para instalar las dependencias de un proyecto Vite?

```
npm build
```

```
npm install
```

```
npm start
```

505. ¿Qué comando inicia el servidor de desarrollo en Vite?

```
npm start
```

```
npm serve
```

```
npm run dev
```

506. ¿Cómo se importa React en un archivo de un proyecto con Vite?

```
import React from "react";
```

```
require("react");
```

```
include React;
```

507. ¿Qué comando se usa para agregar una librería (ejemplo: axios) al proyecto?

```
npm create axios
```

```
npm install axios
```

```
npx add axios
```

508. ¿Cómo se importa una librería instalada en el código?

```
require("axios");
```

```
import axios from "axios";
```

```
import "axios";
```

509. ¿Qué comando se usa para generar una versión optimizada para producción?

```
npm run build
```

```
npm package
```

```
npm run prod
```

510. ¿Qué comando se usa para previsualizar la build generada?

```
npm run test
```

```
npm run preview
```

```
npm run dev
```

511. ¿Qué es React Router?

- Una librería para manejar rutas en aplicaciones React
 - Un hook para manejar estado
 - Un servidor web para React
-

512. ¿Cómo se instala React Router?

```
npm install react-router
```

```
npm install react-router-dom
```

```
npm install router-react
```

513. ¿Qué componente envuelve toda la aplicación para habilitar el enrutamiento?

```
<RouterProvider>
```

```
<RoutesWrapper>
```

```
<BrowserRouter>
```

514. ¿Qué componente se usa para definir un conjunto de rutas?

```
<Routes>
```

```
<Switch>
```

```
<RouteList>
```

515. ¿Qué componente se usa para declarar una ruta?

```
<Route>
```

```
<LinkRoute>
```

```
<Path>
```

516. ¿Cómo se define una ruta para `/about` que renderiza el componente `About` ?

```
<Route url="/about"><About /></Route>
```

```
<Route component={About} path="/about" />
```

```
<Route path="/about" element={<About />} />
```

517. ¿Qué componente se usa para crear enlaces de navegación declarativos?

<Navigate>

<a>

<Link>

518. ¿Qué imprime este código si la URL es /user/42 ?

```
function User() {  
  const params = useParams();  
  return <p>{params.id}</p>;  
}
```

- undefined
 - 42
 - user
-

519. ¿Qué ocurre si se cambia el valor del contexto en el Provider ?

- Los componentes que usan useContext se re-renderizan con el nuevo valor
 - Nada cambia hasta reiniciar la aplicación
 - Lanza un error
-

531. ¿Qué componente se usa para navegación declarativa?

<Link>

```
<Navigate>
```

```
<a>
```

532. ¿Qué imprime este código?

```
<Link to="/about">Acerca de</Link>
```

- Un enlace que navega a /about
- Un botón
- Nada

533. ¿Qué Hook se usa para la navegación programática en React Router?

- useHistory
- useNavigate
- useLocation

534. ¿Qué ocurre al ejecutar este código?

```
const navigate = useNavigate();
navigate("/home");
```

- Redirige a la ruta /home
- Muestra un error
- No hace nada

535. ¿Cuál es la diferencia entre <Link> y <a> en React Router?

- <Link> solo funciona en HTML puro
- <Link> evita la recarga completa de la página, <a> recarga el documento

No hay diferencia

536. ¿Qué componente se usa para redirigir automáticamente a otra ruta?

- `<Redirect to="/ruta" />`
 - `<RouterRedirect to="/ruta" />`
 - `<Navigate to="/ruta" />`
-

537. ¿Qué imprime este código si la ruta actual es `/`?

```
<Navigate to="/login" />
```

- Muestra `/`
 - Nada
 - Redirige automáticamente a `/login`
-

538. ¿Qué Hook se puede usar para ir atrás en la navegación?

```
const navigate = useNavigate();
navigate(-1);
```

```
const back = useHistory();
back();
```

```
const goBack = useRouter();
goBack();
```

539. ¿Qué ocurre al hacer clic en este código?

```
<Link to="/">Inicio</Link>
```

- Muestra un error
 - No ocurre nada
 - Navega a la ruta raíz /
-

540. ¿Qué ventaja tiene la navegación con React Router?

- Siempre recarga la página para asegurar cambios
 - No recarga la página completa, actualiza solo la vista necesaria
 - Solo funciona con backend en Node.js
-

400. ¿En qué parte del documento HTML se recomienda colocar la referencia al archivo CSS externo?

- Al final del documento
 - Dentro de <head>
 - Dentro de <body>
-

401. ¿Qué etiqueta se usa para integrar JavaScript en HTML?

- <javascript>
 - <code>
 - <script>
-

402. ¿Cómo se escribe un script interno en HTML?

-

```
<javascript>
  console.log("Hola mundo");
</javascript>
```

-

```
<code>
  console.log("Hola mundo");
</code>
```

```
<script>
  console.log("Hola mundo");
</script>
```

403. ¿Cómo se enlaza un archivo externo de JavaScript?

```
<link rel="script" href="app.js">
```

```
<script src="app.js"></script>
```

```
<js src="app.js"></js>
```

404. ¿Dónde se recomienda colocar los scripts en un documento HTML para optimizar la carga?

- En cualquier parte del documento
- Justo antes de cerrar la etiqueta `</body>`
- Al inicio dentro de `<head>`

405. ¿Qué atributo del script permite cargar el archivo JavaScript de forma asíncrona?

- `async`
- `defer`
- `autoload`

406. ¿Qué atributo asegura que el script se ejecute después de que el HTML esté parseado?

- `defer`
 - `delay`
 - `async`
-

407. ¿Qué imprime este código si se integra en una página?

```
<script>
  alert("Bienvenido");
</script>
```

- Una alerta con el texto "Bienvenido"
 - Un mensaje en la consola
 - Nada
-

408. ¿Qué ocurre si se enlaza un archivo JS inexistente en `<script src="archivo.js">` ?

- El navegador ignora la etiqueta
 - Se produce un error en la consola pero la página sigue cargando
 - La página no carga en absoluto
-

409. ¿Qué ventaja tiene usar archivos externos de JavaScript en lugar de scripts internos?

- No requieren navegador moderno
 - Permiten reutilizar el código en múltiples páginas
 - Son más rápidos que el código interno
-

410. ¿Qué imprime este código en la consola?

```
<script>
  console.log("Hola desde JS");
</script>
```

- Nada
 - Una alerta con el mensaje
 - Hola desde JS
-

411. ¿Qué es React?

- Un lenguaje de programación nuevo
 - Un sistema operativo
 - Una librería de JavaScript para construir interfaces de usuario
-

412. ¿Quién desarrolla y mantiene React?

- Google
 - Microsoft
 - Facebook (ahora Meta)
-

413. ¿Cuál es el enfoque principal de React?

- Administración de bases de datos
 - Desarrollo de sistemas operativos
 - Construcción de interfaces de usuario basadas en componentes
-

414. ¿Cómo se define React dentro del ecosistema de desarrollo?

- Un compilador
 - Una librería
 - Un framework completo
-

415. ¿En qué lenguaje está basado React?

- JavaScript
 - Python
 - PHP
-

†16. ¿Qué significa que React use un enfoque **declarativo**?

- El programador debe manipular el DOM manualmente
 - Se describe el resultado esperado de la interfaz y React se encarga de actualizar el DOM
 - Todo se escribe en lenguaje SQL
-

†17. ¿Qué caracteriza a la programación **reactiva**?

- El programador debe refrescar manualmente la pantalla
 - La interfaz responde automáticamente a los cambios en el estado o datos
 - Los datos nunca cambian en el tiempo
-

†18. ¿Qué ventaja tiene el enfoque declarativo en React?

- Hace el código más predecible y fácil de entender
 - Permite manipular directamente el DOM con `document.getElementById`
 - Requiere más líneas de código que el enfoque imperativo
-

†19. ¿Qué ocurre cuando cambia el estado en un componente de React?

- El programador debe llamar manualmente a `render()`
 - No ocurre nada hasta reiniciar la aplicación
 - La interfaz se actualiza automáticamente (reactividad)
-

†20. ¿Cuál de las siguientes es una característica de la **programación reactiva** en React?

- Se deben usar funciones externas para refrescar la interfaz
 - Solo se renderiza una vez y luego nunca cambia
 - Los componentes se vuelven a renderizar automáticamente al cambiar el estado o props
-

†21. ¿Qué es un componente en React?

- Un archivo CSS para estilos
- Una base de datos para guardar información

- Una función o clase que devuelve elementos de interfaz de usuario
-

422. ¿Cómo se declara un **componente funcional** básico en React?

-

```
function Hola() {  
  return <h1>Hola</h1>;  
}
```

-

```
let Hola = <h1>Hola</h1>;
```

-

```
function Hola {  
  <h1>Hola</h1>  
}
```

423. ¿Cómo se usan los componentes en React dentro de JSX?

-

```
Componente();
```

-

```
{Componente}
```

-

```
<Componente />
```

424. ¿Cuál es la forma correcta de declarar un componente como **arrow function**?



```
const Hola => <h1>Hola</h1>;
```



```
const Hola = () => <h1>Hola</h1>;
```



```
const Hola = () => <h1>Hola</h1>;
```

425. ¿Qué distingue a un **componente de clase**?

- Solo puede devolver cadenas de texto
- Usa la palabra clave `class` y extiende `React.Component`
- No puede tener estado

426. ¿Cuál es la sintaxis básica de un **componente de clase**?



```
class Hola {  
  return <h1>Hola</h1>;  
}
```



```
class Hola extends React.Component {  
  render() {  
    return <h1>Hola</h1>;  
  }  
}
```



```
function class Hola() {  
  return <h1>Hola</h1>;  
}
```

427. ¿Qué característica tienen los **componentes reutilizables**?

- Pueden usarse varias veces en diferentes partes de la aplicación
 - No aceptan parámetros
 - Solo se pueden declarar una vez
-

428. ¿Cómo se pueden pasar datos a un componente?

- Modificando el DOM manualmente
 - Usando **props**
 - Usando directamente variables globales
-

429. ¿Qué imprime este código?

```
function Saludo(props) {  
  return <h1>Hola {props.nombre}</h1>;  
}  
<Saludo nombre="Ana" />
```

- Hola {props.nombre}
 - Hola
 - Hola Ana
-

430. ¿Cuál es la ventaja principal de usar componentes en React?

- Reutilización y modularidad del código
 - Permiten usar Java en lugar de JavaScript
 - Se necesita menos HTML
-

431. ¿Cómo se pasan parámetros a un componente en React?

- A través de **props**
 - Modificando directamente el DOM
 - Usando variables globales
-

432. ¿Qué imprime este código?

```
function Saludo(props) {  
  return <h1>Hola {props.nombre}</h1>;  
}  
  
<Saludo nombre="Ana" />
```

- Hola
 - Hola {props.nombre}
 - Hola Ana
-

433. ¿Cómo se define un evento `onClick` en React?

-

```
<button onClick="miFuncion()">Click</button>
```

-

```
<button onclick="miFuncion()">Click</button>
```

-

```
<button onClick={miFuncion}>Click</button>
```

434. ¿Qué imprime este código?

```
function Boton() {  
  function manejarClick() {  
    console.log("Clickeado");  
  }  
  return <button onClick={manejarClick}>Click</button>;  
}
```

- "Clickeado" inmediatamente
 - "Clickeado" al hacer click
 - Error
-

435. ¿Cómo se pasa un parámetro a un evento en React?

-

```
<button onclick="miFuncion('Hola')">  
  Click  
</button>
```

-

```
<button onClick={miFuncion("Hola")}>  
  Click  
</button>
```

-

```
<button onClick={() => miFuncion("Hola")}>  
  Click  
</button>
```

436. ¿Qué imprime este código?

```
function Mostrar(props) {  
  return <p>{props.mensaje}</p>;  
}  
<Mostrar mensaje="Hola React" />
```

- mensaje
 - Hola React
 - undefined
-

437. ¿Qué ocurre si se pasa una función como prop?

- Puede ejecutarse dentro del componente hijo
 - Se convierte en string automáticamente
 - Lanza un error
-

438. ¿Qué imprime este código?

```
function Hijo(props) {  
  return <button onClick={props.onSaludar}>Saludar</button>;  
}  
function Padre() {  
  function saludar() { console.log("Hola desde padre"); }  
  return <Hijo onSaludar={saludar} />;  
}
```

- "Hola desde padre" al hacer click
 - Error
 - Nada ocurre
-

439. ¿Cómo se pueden pasar múltiples props a un componente?

-

```
<Componente (nombre="Ana", edad=25) />
```

-

```
<Componente {nombre: "Ana", edad: 25} />
```

-

```
<Componente nombre="Ana" edad={25} />
```

440. ¿Cuál es la forma correcta de manejar eventos en React?

- Usar HTML puro sin eventos
 - Usar camelCase (`onClick`, `onChange`) y pasar funciones
 - Usar atributos en minúsculas (`onclick`) y pasar strings
-

441. ¿Qué son los Hooks en React?

- Funciones especiales que permiten usar estado y otras características en componentes funcionales
 - Un sistema de estilos en React
 - Una nueva forma de declarar clases en React
-

442. ¿En qué versión de React se introdujeron los Hooks?

- React 16.8
 - React 18
 - React 15.0
-

443. ¿Cuál es la regla principal al usar Hooks?

- Pueden declararse dentro de cualquier función
 - Pueden usarse dentro de cualquier bucle o condicional
 - Solo se pueden usar en el nivel superior de un componente
-

444. ¿Cuál es un Hook básico para manejar el estado?

- `useState`
 - `useEffect`
 - `useContext`
-

145. ¿Qué Hook se usa para manejar efectos secundarios (ejemplo: llamadas a APIs)?

- `useEffect`
 - `useState`
 - `useReducer`
-

146. ¿Qué permite hacer el Hook `useContext` ?

- Crear un nuevo estado local
 - Definir un ciclo de vida
 - Acceder a un contexto sin necesidad de usar un Consumer
-

147. ¿Cuál es la ventaja principal de los Hooks?

- Reemplazan totalmente a los componentes de clase
 - Permiten usar estado y lógica reutilizable en componentes funcionales
 - Reducen el tamaño de los archivos CSS
-

148. ¿Qué Hook se usa para manejar un valor mutable que no causa renderizados?

- `useEffect`
 - `useRef`
 - `useMemo`
-

149. ¿Qué significa que los Hooks sean **composables**?

- Deben ejecutarse una única vez en la aplicación
 - Se pueden crear Hooks personalizados a partir de otros Hooks
 - Solo se pueden usar dentro de clases
-

150. ¿Qué ocurre si se usan Hooks dentro de un condicional?

- Puede romper las reglas de los Hooks y causar errores
- Se convierten en funciones normales

- Funciona sin problemas siempre
-

451. ¿Qué es `useState` ?

- Un Hook que permite añadir estado a los componentes funcionales
 - Una función para renderizar componentes
 - Un método exclusivo de clases
-

452. ¿Cómo se importa `useState` en un componente?

-

```
import { useState } from "react";
```

-

```
import useState from "react";
```

-

```
const useState = require("react");
```

453. ¿Qué devuelve la llamada a `useState` ?

- Un array con el valor actual y una función para actualizarlo
 - Solo el valor inicial
 - Un objeto con el estado y un método `setState`
-

454. ¿Qué imprime este código?

```
const [contador, setContador] = useState(0);
console.log(contador);
```

- `undefined`

- 0
 - Error
-

455. ¿Cómo se actualiza el estado con `useState` ?

- Reiniciando el componente
 - Modificando directamente la variable
 - Usando la función de actualización (`setAlgo`)
-

456. ¿Qué imprime este código después de un click?

```
const [contador, setContador] = useState(0);
<button onClick={() => setContador(contador + 1)}>Incrementar</button>
```

- El valor de `contador` aumenta en 1
 - Da error
 - Siempre imprime 0
-

457. ¿Qué ocurre si se llama a `setEstado` varias veces seguidas?

- Ejecuta cada una inmediatamente en orden
 - React agrupa las actualizaciones para optimizar
 - Lanza un error
-

458. ¿Cuál es la sintaxis correcta para usar un valor inicial dinámico en `useState` ?

-

```
const [valor, setValor] = useState = calcularInicial();
```

-

```
const [valor, setValor] = useState(() => calcularInicial());
```

-

```
const [valor, setValor] = useState{calcularInicial()};
```

459. ¿Qué pasa si se actualiza el estado con el mismo valor?

- Lanza un error
 - Siempre vuelve a renderizar
 - React no vuelve a renderizar el componente
-

460. ¿Qué imprime este código?

```
const [texto, setTexto] = useState("Hola");
setTexto("Mundo");
console.log(texto);
```

- Error
 - "Mundo" inmediatamente
 - "Hola" en el render actual
-

461. ¿Cómo se actualiza correctamente un objeto en `useState` ?

-

```
setEstado({ ...estado, nuevoValor: 123 });
```

-

```
setEstado(estado.nuevoValor = 123);
```

-

```
estado.nuevoValor = 123;
```

462. ¿Qué ocurre si se modifica directamente un objeto en `useState` ?

- React actualiza automáticamente
 - Se lanza un error
 - No se re-renderiza el componente
-

463. ¿Cómo se agrega un elemento a un array en el estado?

```
setLista([...lista, nuevoElemento]);
```

```
lista.push(nuevoElemento);
```

```
setLista(lista.push(nuevoElemento));
```

464. ¿Qué imprime este código?

```
const [usuario, setUsuario] = useState({ nombre: "Ana", edad: 20 });
setUsuario({ ...usuario, edad: 21 });
console.log(usuario.edad);
```

- 21
 - 20
 - Error
-

465. ¿Cómo se elimina un elemento de un array en el estado?

```
delete lista[valor];
```

```
setLista(lista.filter(item => item !== valor));
```

```
lista.remove(valor);
```

466. ¿Cómo se actualiza una propiedad anidada en un objeto con `useState` ?

```
setEstado({ ...estado, direccion: { ...estado.direccion, ciudad: "Roma" } });
```

```
estado.direccion.ciudad = "Roma";
```

```
setEstado(estado.direccion.ciudad = "Roma");
```

467. ¿Qué ocurre si se reemplaza el array en `useState` sin usar spread?

- Se pierde el contenido anterior
 - Lanza un error
 - Se conservan los elementos previos automáticamente
-

468. ¿Cómo se agrega un objeto nuevo a un array en el estado?

```
setUsuarios(usuarios.add({ id: 3, nombre: "Luis" }));
```

```
usuarios.push({ id: 3, nombre: "Luis" });
```

```
setUsuarios([...usuarios, { id: 3, nombre: "Luis" }]);
```

469. ¿Qué pasa si se llama a `setEstado` con el mismo objeto sin crear una copia?

- React siempre vuelve a renderizar
- React no detecta cambios y no renderiza de nuevo
- Da un error

470. ¿Qué imprime este código?

```
const [lista, setLista] = useState([1, 2]);
setLista([...lista, 3]);
console.log(lista);
```

- [1, 2, 3]
- [1, 2]
- Error

471. ¿Qué es `useEffect` en React?

- Un Hook que maneja efectos secundarios en componentes funcionales
- Un método exclusivo de clases
- Una función para actualizar el estado

472. ¿Cómo se importa `useEffect`?

```
import useEffect from "react";
```

```
const useEffect = require("react");
```

```
import { useEffect } from "react";
```

473. ¿Qué imprime este código?

```
useEffect(() => {
  console.log("Efecto ejecutado");
});
```

- Nunca imprime nada
 - "Efecto ejecutado" solo en el primer render
 - "Efecto ejecutado" en cada render
-

474. ¿Cómo se ejecuta un efecto solo una vez al montar el componente?

- Pasando un array vacío como dependencia: `useEffect(() => {...}, [])`
 - Usando `useEffect(() => {...})` sin dependencias
 - Usando `useEffectOnce`
-

475. ¿Qué ocurre si se pasa un array con dependencias a `useEffect` ?

- El efecto nunca se ejecuta
 - El efecto se ejecuta siempre en cada render
 - El efecto se ejecuta cuando cambian esas dependencias
-

476. ¿Qué imprime este código?

```
const [contador, setContador] = useState(0);
useEffect(() => {
  console.log("Cambio en contador");
}, [contador]);
```

-
- Nunca imprime nada
 - "Cambio en contador" solo una vez
 - "Cambio en contador" cada vez que cambia contador

477. ¿Cómo se limpia un efecto en useEffect ?

- Usando useCleanup
 - Retornando una función de limpieza dentro del callback
 - Llamando a clearEffect()
-

478. ¿Qué ocurre con este código?

```
useEffect(() => {
  const id = setInterval(() => console.log("Tick"), 1000);
  return () => clearInterval(id);
}, []);
```

- Da un error
 - Imprime "Tick" una vez y se detiene
 - Imprime "Tick" cada segundo y se limpia al desmontar el componente
-

479. ¿Qué problema puede ocurrir si no se limpian los efectos correctamente?

- El estado deja de actualizarse
 - Fugas de memoria o comportamientos inesperados
 - El componente no se renderiza
-

480. ¿Qué imprime este código?

```
const [x, setX] = useState(0);
useEffect(() => {
  console.log("Render con x:", x);
}, [x]);
setX(5);
```

- Error
 - Solo "Render con x: 0"
 - "Render con x: 0" y luego "Render con x: 5"
-

481. ¿Qué es `useContext` en React?

- Un Hook que permite acceder a un contexto sin necesidad de usar un Consumer
 - Un método exclusivo de clases
 - Una función para manejar efectos secundarios
-

482. ¿Cómo se importa `useContext` ?

-

```
import useContext from "react";
```

-

```
const useContext = require("react");
```

-

```
import { useContext } from "react";
```

483. ¿Qué se necesita antes de usar `useContext` ?

- Definir un `useState`
 - Crear un contexto con `React.createContext()`
 - Importar un reducer
-

484. ¿Cómo se accede a un valor de contexto con `useContext` ?

```
const valor = useContext(MiContexto);
```

```
const valor = MiContexto();
```

```
const valor = getContext(MiContexto);
```

485. ¿Qué imprime este código?

```
const TemaContext = React.createContext("claro");
function Componente() {
  const tema = useContext(TemaContext);
  return <p>{tema}</p>;
}
```

- `undefined`
- Error
- `"claro"`

486. ¿Qué ocurre si un componente no está envuelto en un `Provider` ?

- Obtiene el valor por defecto del contexto
- Obtiene `null`
- Siempre da error

487. ¿Cómo se define un `Provider` de contexto?

```
<MiContexto value="algo">
  <App />
</MiContexto>
```

```
<Provider contexto={MiContexto} valor="algo">
  <App />
</Provider>
```

```
<MiContexto.Provider value="algo">
  <App />
</MiContexto.Provider>
```

488. ¿Qué imprime este código?

```
const UsuarioContext = React.createContext("Invitado");
function Perfil() {
  const usuario = useContext(UsuarioContext);
  return <h1>{usuario}</h1>;
}
<UsuarioContext.Provider value="Ana">
  <Perfil />
</UsuarioContext.Provider>
```

- "Invitado"
 - undefined
 - "Ana"
-

489. ¿Qué ventaja tiene `useContext` ?

- Permite compartir datos globales sin necesidad de pasar props manualmente
 - Reemplaza a `useState`
 - Hace que los componentes nunca se re-rendericen
-

490. ¿Qué ocurre si cambia el valor del contexto en el `Provider` ?

- Nada cambia hasta reiniciar la aplicación
 - Lanza un error
 - Los componentes que usan `useContext` se re-renderizan con el nuevo valor
-

491. ¿Qué es `useRef` en React?

- Un método exclusivo de clases
 - Una función para crear estado
 - Un Hook que devuelve un objeto mutable con la propiedad `.current`
-

492. ¿Cómo se importa `useRef` ?

-

```
const useRef = require("react");
```

-

```
import { useRef } from "react";
```

-

```
import useRef from "react";
```

493. ¿Qué valor inicial tiene una referencia creada con `useRef()` ?

- `undefined`
 - `null`
 - `0`
-

494. ¿Qué imprime este código?

```
const ref = useRef(5);
console.log(ref.current);
```

- Error
 - undefined
 - 5
-

495. ¿Qué ocurre cuando se actualiza `.current` de un `useRef` ?

- No causa re-render del componente
 - Lanza un error
 - Siempre vuelve a renderizar el componente
-

496. ¿Cuál es un uso común de `useRef` ?

- Definir reducers
 - Referenciar elementos del DOM
 - Crear un estado global
-

497. ¿Qué imprime este código?

```
function Componente() {
  const inputRef = useRef(null);
  return <input ref={inputRef} />;
}
```

- Error
 - Una referencia al elemento `<input>`
 - null
-

498. ¿Qué ocurre si se cambia manualmente `ref.current` ?

- Lanza un error
- Se vuelve a renderizar automáticamente
- Se actualiza el valor pero no se renderiza el componente

499. ¿Cómo se usa `useRef` para almacenar un valor que persiste entre renders sin causar renderizados?

```
const contador = useRef(0);  
contador.current += 1;
```

```
const [contador, setContador] = useRef(0);
```

```
const contador = useRefState(0);
```

500. ¿Qué imprime este código?

```
const ref = useRef("Hola");  
ref.current = "Mundo";  
console.log(ref.current);
```

- `undefined`
 - `"Mundo"`
 - `"Hola"`
-

501. ¿Qué es Vite?

- Una herramienta de construcción rápida para proyectos frontend
 - Una base de datos
 - Un framework de backend
-

502. ¿Cómo se crea un nuevo proyecto con Vite usando npm?

```
npm create vite@latest
```

```
npx start vite
```

```
npm init react-app
```

503. ¿Qué comando se usa para entrar en la carpeta del proyecto recién creado?

```
cd nombre-proyecto
```

```
npm cd nombre-proyecto
```

```
cd vite
```

504. ¿Qué comando se usa para instalar las dependencias de un proyecto Vite?

```
npm build
```

```
npm install
```

```
npm start
```

505. ¿Qué comando inicia el servidor de desarrollo en Vite?

```
npm serve
```

```
npm run dev
```

```
npm start
```

506. ¿Cómo se importa React en un archivo de un proyecto con Vite?

```
require("react");
```

```
import React from "react";
```

```
include React;
```

507. ¿Qué comando se usa para agregar una librería (ejemplo: axios) al proyecto?

```
npm install axios
```



```
npm create axios
```



```
vite add axios
```

508. ¿Cómo se importa una librería instalada en el código?



```
require("axios");
```



```
import axios from "axios";
```



```
import "axios";
```

509. ¿Qué comando se usa para generar una versión optimizada para producción?



```
npm package
```



```
npm run build
```



```
npm run prod
```

510. ¿Qué comando se usa para previsualizar la build generada?

```
npm run dev
```

```
npm run preview
```

```
npm run test
```

511. ¿Qué es React Router?

- Un hook para manejar estado
 - Una librería para manejar rutas en aplicaciones React
 - Un servidor web para React
-

512. ¿Cómo se instala React Router?

```
npm install router-react
```

```
npm install react-router-dom
```

```
npm install react-router
```

513. ¿Qué componente envuelve toda la aplicación para habilitar el enrutamiento?

```
<RouterProvider>
```

```
<RoutesWrapper>
```

```
<BrowserRouter>
```

514. ¿Qué componente se usa para definir un conjunto de rutas?

```
<RouteList>
```

```
<Switch>
```

```
<Routes>
```

515. ¿Qué componente se usa para declarar una ruta?

<Route>

<Path>

<LinkRoute>

516. ¿Cómo se define una ruta para /about que renderiza el componente About ?

<Route path="/about" element={<About />} />

<Route url="/about"><About /></Route>

<Route component={About} path="/about" />

517. ¿Qué componente se usa para crear enlaces de navegación declarativos?

<Navigate>

<Link>

```
<a>
```

518. ¿Qué imprime este código?

```
<Link to="/home">Inicio</Link>
```

- Un enlace que navega a `/home`
- Un `div` con texto
- Un botón que cambia el estado

519. ¿Qué ocurre si ninguna ruta coincide con la URL?

- React Router genera automáticamente un error 404
- La aplicación se cierra
- Se debe declarar una ruta "catch-all" (`*`) para manejar páginas no encontradas

520. ¿Cuál es la ventaja del enfoque declarativo en React Router?

- El enrutamiento se describe con JSX, fácil de entender y mantener
- Solo funciona con clases, no con funciones

```
### parámetros dinámicos en React Router
```

- Obliga a manipular manualmente el `window.location`

521. ¿Cómo se define un parámetro dinámico en una ruta?

- Usando `$param` en la propiedad `path`
- Usando `:nombreParametro` en la propiedad `path`
- Usando `{param}` en la propiedad `path`

522. ¿Cuál es la forma correcta de definir una ruta dinámica para `/user/:id` ?

```
<Route path="/user/$id" element={<User />} />
```

```
<Route path="/user/{id}" element={<User />} />
```

```
<Route path="/user/:id" element={<User />} />
```

523. ¿Qué Hook se usa para acceder a los parámetros dinámicos en un componente?

- useNavigate
 - useParams
 - useLocation
-

524. ¿Qué imprime este código si la URL es `/user/42` ?

```
function User() {  
  const params = useParams();  
  return <p>{params.id}</p>;  
}
```

- 42
 - user
 - undefined
-

525. ¿Cómo se pueden obtener múltiples parámetros dinámicos?

- Definiendo varios en la ruta: `/post/:id/:comentarioId`
 - Concatenando parámetros con `?`
 - Usando `props` directamente
-

526. ¿Qué imprime este código si la URL es /post/10/5 ?

```
function Post() {  
  const { id, comentarioId } = useParams();  
  return <p>{id} - {comentarioId}</p>;  
}
```

- post - comentario
 - undefined - undefined
 - 10 - 5
-

527. ¿Qué ocurre si no se define un parámetro dinámico en la URL?

- La aplicación no renderiza
 - useParams devuelve undefined para ese parámetro
 - React Router lanza un error
-

528. ¿Qué diferencia hay entre parámetros dinámicos y query params?

- Los dinámicos se definen en la ruta con :param , los query params se pasan con ?
 - Los dinámicos se definen en el estado
 - Son lo mismo
-

529. ¿Cómo se acceden a los query params en React Router?

- Usando useParams
 - Usando useLocation y URLSearchParams
 - Usando useQuery
-

530. ¿Qué imprime este código si la URL es /search?q=react ?

```
function Search() {  
  const { search } = useLocation();  
  const query = new URLSearchParams(search);  
  return <p>{query.get("q")}</p>;  
}
```

- react
 - q=react
 - undefined
-

531. ¿Qué componente se usa para navegación declarativa?

- <Link>
 - <a>
 - <Navigate>
-

532. ¿Qué imprime este código?

```
<Link to="/about">Acerca de</Link>
```

- Un botón
 - Un enlace que navega a /about
 - Nada
-

533. ¿Qué Hook se usa para la navegación programática en React Router?

- useHistory
 - useLocation
 - useNavigate
-

534. ¿Qué ocurre al ejecutar este código?

```
const navigate = useNavigate();  
navigate("/home");
```

- Muestra un error
 - Redirige a la ruta `/home`
 - No hace nada
-

535. ¿Cuál es la diferencia entre `<Link>` y `<a>` en React Router?

- No hay diferencia
 - `<Link>` solo funciona en HTML puro
 - `<Link>` evita la recarga completa de la página, `<a>` recarga el documento
-

536. ¿Qué componente se usa para redirigir automáticamente a otra ruta?

```
<RouterRedirect to="/ruta" />
```

```
<Navigate to="/ruta" />
```

```
<Redirect to="/ruta" />
```

537. ¿Qué imprime este código si la ruta actual es `/`?

```
<Navigate to="/login" />
```

- Muestra `/`
 - Nada
 - Redirige automáticamente a `/login`
-

538. ¿Qué Hook se puede usar para ir atrás en la navegación?



```
const back = useHistory();
back();
```



```
const navigate = useNavigate();
navigate(-1);
```



```
const goBack = useRouter();
goBack();
```

539. ¿Qué ocurre al hacer clic en este código?

```
<Link to="/">Inicio</Link>
```

- Muestra un error
 - Navega a la ruta raíz /
 - No ocurre nada
-

540. ¿Qué ventaja tiene la navegación con React Router?

- Solo funciona con backend en Node.js
 - No recarga la página completa, actualiza solo la vista necesaria
 - Siempre recarga la página para asegurar cambios
-

541. ¿Qué es React Hook Form?

- Una librería para manejar formularios en React con Hooks
 - Un nuevo Hook de React
 - Una librería para manejar rutas
-

542. ¿Cuál es el principal objetivo de React Hook Form?

- Conectar React con bases de datos
 - Facilitar la validación y manejo de formularios
 - Manejar animaciones en React
-

543. ¿Cómo se instala React Hook Form?

-

```
npm install hook-form
```

-

```
npm install react-hook-form
```

-

```
npm install form-react
```

544. ¿Qué Hook principal provee la librería?

- useForm
 - useState
 - useEffect
-

545. ¿Qué devuelve useForm ?

- Un objeto con métodos para manejar el formulario (register , handleSubmit , etc.)
 - Solo un array con inputs
 - Una función que renderiza un formulario
-

546. ¿Cómo se registra un input en React Hook Form?

- Usando useInput

- Usando `useState`
 - Usando la función `register`
-

547. ¿Qué imprime este código si se envía el formulario?

```
const { register, handleSubmit } = useForm();
const onSubmit = data => console.log(data);
<form onSubmit={handleSubmit(onSubmit)}>
  <input {...register("nombre")}>
  <button type="submit">Enviar</button>
</form>
```

- `undefined`
 - Un objeto con `{ nombre: "valorIngresado" }`
 - Solo el string `"valorIngresado"`
-

548. ¿Qué ocurre si no se envuelve el `onSubmit` con `handleSubmit`?

- El formulario no maneja los datos correctamente
 - Se envía igual
 - Lanza un error automático
-

549. ¿Cómo se acceden a los errores de validación?

- Desde `formState.errors`
 - Usando `useErrors`
 - Con `errorMessage()`
-

550. ¿Qué Hook se usa para observar valores del formulario en tiempo real?

- `listen`
 - `observe`
 - `watch`
-

551. ¿Cómo se agrega una validación requerida en un input?

```
<input required {...register("email")}>
```

```
<input register="email" />
```

```
<input {...register("email", { required: true })}>
```

552. ¿Qué imprime este código si el campo está vacío y se intenta enviar?

```
const { register, handleSubmit, formState: { errors } } = useForm();
<form onSubmit={handleSubmit(data => console.log(data))}>
  <input {...register("nombre", { required: true })}>
  {errors.nombre && <span>Requerido</span>}
  <button type="submit">Enviar</button>
</form>
```

- Muestra "Requerido"
 - Envía los datos vacíos
 - Muestra "undefined"
-

553. ¿Cómo se define una validación de longitud mínima?

- { minlength: 5 }
 - { length: { min: 5 } }
 - { minLength: 5 }
-

554. ¿Cómo se valida un campo con expresión regular?

- { regex: /^[A-Za-z]+\$/ }

- { pattern: /^[A-Za-z]+\$/ }
 - { match: /^[A-Za-z]+\$/ }
-

555. ¿Qué Hook se usa para resetear valores de un formulario?

- useReset
 - clear
 - reset
-

556. ¿Cómo se establece un valor inicial en `useForm`?

- - ```
const { register } = useForm("Ana");
```
  - 
  - ```
const { register } = useForm({ initialValues: { nombre: "Ana" } });
```
 -
 - ```
const { register } = useForm({ initialValues: { nombre: "Ana" } });
```
- 

557. ¿Cómo se maneja un formulario con varios campos?

- Pasando todos en un array
  - Registrando cada campo con `register("campo")`
  - Usando un único campo global
- 

558. ¿Qué imprime este código al enviar?

```
const { register, handleSubmit } = useForm();
const onSubmit = data => console.log(data);
<form onSubmit={handleSubmit(onSubmit)}>
 <input {...register("usuario")}>
 <input {...register("email")}>
 <button type="submit">Enviar</button>
</form>
```

- Solo el último valor ingresado
  - undefined
  - Un objeto con { usuario: "valor", email: "valor" }
- 

559. ¿Qué ventaja tiene React Hook Form sobre usar `useState` para inputs?

- Permite usar CSS automáticamente
  - Hace que los inputs nunca cambien
  - Mejora el rendimiento al no causar renders innecesarios
- 

560. ¿Qué ocurre si se llama a `reset()`?

- Lanza un error
  - Los campos del formulario vuelven a sus valores iniciales
  - Borra el formulario del DOM
- 

561. ¿Qué es shadcn/ui?

- Una librería de animaciones
  - Un framework backend para Node.js
  - Una colección de componentes reutilizables para React construidos con TailwindCSS
- 

562. ¿Cuál es el objetivo principal de shadcn/ui?

- Proporcionar componentes accesibles y estilizados listos para usar en proyectos React
- Sustituir completamente a React
- Manejar el enrutamiento en aplicaciones

---

563. ¿Con qué librería de estilos trabaja principalmente shadcn/ui?

- Material UI
  - TailwindCSS
  - Bootstrap
- 

564. ¿Qué filosofía sigue shadcn/ui en lugar de ser una librería publicada en npm?

- Se instalan solo como paquete binario
  - Los componentes se copian directamente al proyecto, dando control total al desarrollador
  - Requiere siempre conexión a internet para usarse
- 

565. ¿Qué comando se usa para instalar shadcn/ui en un proyecto con React y Tailwind?

```
npx shadcn-ui init
```

```
npm install shadcn-ui
```

```
npx create-shadcn
```

---

566. ¿Qué archivo se genera tras inicializar shadcn/ui?

- ui.config.json
  - components.json
  - shadcn.config.js
- 

567. ¿Qué comando se usa para agregar un componente específico de shadcn/ui?

```
npx shadcn-ui add button
```

```
npx add shadcn button
```

```
npm install @shadcn/button
```

---

568. ¿Qué requisito es necesario para usar shadcn/ui?

- Tener TailwindCSS configurado en el proyecto
  - Tener Express instalado
  - Usar TypeScript obligatoriamente
- 

569. ¿Qué ventaja tiene copiar los componentes al proyecto en lugar de importarlos como librería?

- Permite personalizarlos libremente sin depender de actualizaciones externas
  - Reduce el tamaño del código
  - Hace que el proyecto funcione sin React
- 

570. ¿Dónde suelen guardarse los componentes agregados de shadcn/ui?

- En la carpeta /lib/shadcn
  - En la carpeta /components
  - En la carpeta /src/hooks
- 

571. ¿Cómo se importa un componente agregado con shadcn/ui?

```
import Button from "shadcn-ui";
```

```
import { Button } from "@/components/ui/button";
```

```
require("@shadcn/button");
```

---

572. ¿Qué imprime este código?

```
import { Button } from "@/components/ui/button";
export default function App() {
 return <Button>Click aquí</Button>;
}
```

- Un botón estilizado con Tailwind
  - Nada
  - Un enlace vacío
- 

573. ¿Cómo se puede personalizar el estilo de un componente de shadcn/ui?

- Editando directamente el archivo del componente en `/components/ui`
  - Usando variables globales de React
  - Instalando un tema desde npm
- 

574. ¿Qué librería suele usarse junto con shadcn/ui para gestionar clases condicionales?

- `axios`
  - `clsx`
  - `lodash`
- 

575. ¿Qué imprime este código si se usa el componente `Input`?

```
import { Input } from "@/components/ui/input";
<Input placeholder="Escribe aquí" />
```

- Un botón
  - Nada
  - Un campo de texto estilizado
- 

576. ¿Qué ventaja tiene shadcn/ui respecto a otras librerías como Material UI?

- No requiere React
  - No necesita Tailwind
  - Da control total sobre el código de los componentes porque están dentro del proyecto
- 

577. ¿Qué ocurre si se borra un componente agregado con shadcn/ui?

- El proyecto deja de compilar
  - Se reinstala automáticamente al ejecutar `npm install`
  - Deja de estar disponible hasta volver a importarlo con `npx shadcn-ui add`
- 

578. ¿Qué comando muestra la ayuda y lista los componentes disponibles?

- 

```
npx shadcn-ui list
```

- 

```
npx shadcn-ui help
```

- 

```
npm run shadcn
```

---

579. ¿Qué tipo de componentes incluye shadcn/ui?

- Únicamente hooks de React
  - Botones, inputs, menús, modales, tooltips, etc.
  - Solo componentes de gráficos
- 

580. ¿Cuál es una limitación de shadcn/ui?

- No permite personalización de estilos
  - No es un paquete npm tradicional, requiere copiar componentes al proyecto
  - Solo funciona con Vue
- 

581. ¿Qué es Next.js?

- Un lenguaje de programación
  - Un framework de React para construir aplicaciones web con renderizado híbrido
  - Una librería de animaciones
- 

582. ¿Quién mantiene Next.js?

- Vercel
  - Meta
  - Google
- 

583. ¿Qué ventaja principal tiene Next.js sobre React puro?

- Soporte integrado para renderizado del lado del servidor (SSR)
  - Permite usar Java en lugar de JavaScript
  - Reemplaza completamente a React
- 

584. ¿Qué comando se usa para crear un nuevo proyecto Next.js?

- 

```
npx create-next-app@latest
```

```
npx start next
```

```
npm init react-app
```

---

585. ¿Dónde se definen las páginas en un proyecto Next.js?

- En la carpeta `src`
  - En la carpeta `pages`
  - En la carpeta `views`
- 

586. ¿Qué ocurre al crear un archivo `pages/about.js` ?

- Genera automáticamente la ruta `/about`
  - Requiere configuración manual del router
  - No genera ninguna ruta
- 

587. ¿Qué motor de estilos soporta Next.js de forma nativa?

- Bootstrap únicamente
  - CSS Modules
  - Material UI únicamente
- 

588. ¿Qué comando inicia el servidor de desarrollo en Next.js?

```
npm run dev
```

```
npm serve
```

```
npm start
```

---

589. ¿Qué significa SSG en Next.js?

- Static Site Generation
  - Simple Server Gateway
  - Server Side GraphQL
- 

590. ¿Qué significa SSR en Next.js?

- Server Side Rendering
  - Static Style Rendering
  - Single State Rehydration
- 

591. ¿Qué hook se usa para obtener la ruta actual?

- `useLocation`
  - `useRouter`
  - `usePath`
- 

592. ¿Qué componente se usa para navegación en Next.js?

- `<Link>` de `next/link`
  - `<Navigate>`
  - `<a>` de HTML
- 

593. ¿Qué imprime este código?

```
import Link from "next/link";
<Link href="/contact">Contacto</Link>
```

- 
- Un enlace que navega a `/contact` sin recargar la página
  - Nada
  - Un botón
- 

594. ¿Cómo se definen rutas dinámicas en Next.js?

- Usando `:id` en el nombre del archivo
  - Creando archivos con corchetes, ejemplo: `[id].js`
  - Usando query params solamente
- 

595. ¿Qué función se usa para obtener datos en el servidor en tiempo de compilación?

- `getInitialProps`
  - `getServerSideProps`
  - `getStaticProps`
- 

596. ¿Qué función se usa para obtener datos en cada request (SSR)?

- `fetchData`
  - `getServerSideProps`
  - `getStaticProps`
- 

597. ¿Qué función se usa para generar rutas dinámicas en tiempo de compilación?

- `getStaticPaths`
  - `getRoutes`
  - `getPaths`
- 

598. ¿Qué ocurre si se crea un archivo dentro de `pages/api/` ?

- Se convierte en una página normal

- Da error
  - Se convierte en una API route
- 

599. ¿Qué plataforma está más integrada para desplegar proyectos Next.js?

- Vercel
  - Firebase
  - Heroku
- 

600. ¿Cuál es una ventaja de usar Next.js para SEO?

- Usa automáticamente palabras clave en HTML
  - El contenido se puede renderizar en el servidor, mejorando la indexación
  - No permite modificar etiquetas <meta>
-