What will be the output of following JavaScript code snippets? In case of Error, mention the error in the code.

|  |  |  |
| --- | --- | --- |
| **No.** | **JavaScript Code** | **Output** |
| 1 | let x = [];  x[3] = 23;  console.log(x.length); | 4  Reason: x[0], x[1], and x[2] will be undefined and x[3] will be 23. |
| 2 | let a = [1, 2, 3, 4, 5,6 ,7, 8];  let b = a;  a.push(9);  console.log(b);  a.pop();  console.log(b); | [1, 2, 3, 4, 5,6 ,7, 8, 9]  [1,2,3,4,5,6,7,8]  Reason: let b = a; assigns the reference of the array a to b, meaning both a and b point to the same array in memory. |
| 3 | let fun = (x = 10) => ‘Value of X is ${x}!’;  console.log(fun());  console.log(fun(5)); | Value of X is 10  Value of X is 5  Reason: arrow function has default value for x parameter. |
| 4 | let array = ['1', '2', ‘3’];  array.push('4');  array.shift();  console.log(array);  array.shift();  console.log(array); | ['2', '3', ‘4’]  [‘3’,’4’]  Reason: array.push('4'); adds '4' to the end of the array, making it ['1', '2', '3', ‘4’].  arr.shift(); removes the first element ('1') from the array, leaving ['2', '3', ‘4’]. |
| 5 | let array = [1, 2, 3, 4, 5];  array.pop();  array.unshift(7,8);  console.log(array); | [7,8, 1, 2,3,4]  Reason: array.pop(); removes the last element (5), leaving the array as [1, 2, 3, 4].  array.unshift(7, 8); adds 7 and 8 to the beginning of the array, making it [7, 8, 1, 2, 3, 4]. |
| 6 | let numbers = [5,6,7,8];  numbers.length = 5;  console.log(numbers); | [5, 6, 7, 8, undefined]  Reason: numbers.length = 5 add one more index to array, value of that index will be undefined |
| 7 | let result = [5, 9, 10, 2, 4].map(x => x <= 5);  console.log(result); | [true, false, false, true, true]  Reason: The code uses an arrow function x => x <= 5.  **x =>** : is the parameter along with the arrow notation, x is a single parameter to the function  **x <= 5** : is the body of the function which is a single condition checking if the value of x is less than equal to 5.  Both the following expressions are the same:   * const val = (x) => {x <= 5}; * const val = x => x <= 5;   The map method is called on the array [5, 9, 10, 2, 4]. The map method creates a new array by applying the arrow function to each element of the original array to check if the element is <= 5, stores true if x is less than or equal to 5, and false otherwise in the mew array |
| 8 | let array = [1, 2, 3, 112, 213];  array.sort();  console.log(array); | [1, 112, 213, 2, 3]  Reason: **Array is sorted in alphabetical order.**  First, all numbers are converted to string and then the comparison is made character by character.   * "1" comes first since it starts with 1, which is less than “2”. * Next, "112" comes before "2" because the first character “1” in “112” is smaller than first character of second number which is “2”. So “112” is smaller than “2”. * Then, "213" comes next. It starts with “2”, which is greater than first character of “112”, so “112” is smaller than “213”. But since "213" is compared to "2", the first character “2” of "213" matches the first character of "2". Since both have the first character as 2, we proceed to compare the next character. "213" has a second character “1”, whereas "2" has no second character (or it can be considered as undefined), meaning it effectively becomes less. Thus, "213" is considered less than "2". |
| 9 | let arr = [[], undefined, {}, null];  console.log(arr.map(n => typeof n)); | ['object', 'undefined', 'object', 'object']  Reason: it is a map function that iterates over each element and passes it to the array function. The arrow function returns type of the element passed to it.  Both the following expressions are the same:   * const val = (n) => { typeof n}; * const val = n => typeof n;   Typeof returns following:   * For null: typeof null returns "object", which is a known quirk in JavaScript. * For undefined: typeof undefined returns "undefined". * For {}: typeof {} returns "object", since it's an object. * For []: typeof [] also returns "object", because arrays are a type of object in JavaScript |
| 10 | function val (a=4, b=5) { return a-b }  function val(a=4) { return a\*a }  console.log(val(2, 3)) | 4  Reason: In JavaScript, if you define two functions with the same name in the same scope, **the second definition will override the first one**. In this case, the first function val(a = 4, b = 5) is overridden by the second function val(a = 4). Each time you call val function, a second function will be called.  After that, the val function only takes one parameter (a), and it has a default value of 4. It returns a \* a.  When you call val(2, 3), the first argument 2 is passed as the value of a, and the second argument 3 is ignored because the second parameter was removed in the overriding function.  Inside the function, a is 2, so val returns 2 \* 2, which is 4. |
| 11 | let result = 3 + "4";  console.log(result); | “34”  Reason: 3 is a number.  "4" is a string.  When you use the + operator between a number and a string, JavaScript converts the number to a string and concatenates them.  Therefore, 3 + "4" results in the string "34". |
| 12 | let fruits = ["Apple", "Banana", "Cherry", "Date"];  fruits.splice(1, 2, "Orange", "Grape");  console.log(fruits); | ["Apple", "Orange", "Grape", "Date"]  Reason: **The splice() method is used to change the contents of an array by removing or replacing existing elements and/or adding new elements in place.**  fruits.splice(1, 2, "Orange", "Grape"); // Start at index 1, remove 2 items, which are “Banana” and “Cherry” then two new items "Orange" and "Grape" at their place |
| 12 | let colors = ["Red", "Green", "Blue"];  let colorString = colors.join(", ");  console.log(coloring); | "Red, Green, Blue"  Reason: **The join() method is used to join all elements of an array into a string, with a specified separator between each element.**  In this case, it creates a string of all elements separated by “,”. |
| 13 | function multiply (factor, ...numbers) {  return numbers.map(num => num \* factor);  }  const result = multiply(4, 1, 2, 3);  console.log(result); | [4, 8, 12]  Reason: **Rest operator is used since it can pass an unknown number of arguments to a function.**  The function has 2 parameters.  First parameter which is “factor” takes the first value which is 4.  Second parameter which is “numbers” is a rest operator which takes the remaining values passed which are 1,2,3.  In the function, a map function is applied to the rest parameter array. The map function takes one element at a time from the rest array and multiplies the element with the factor and stores the result in a new array which is returned from the function.  In this case factor value is 4.  Map function does this:  1 \* 4  2 \* 4  3 \* 4  Results 4, 8 and 12 are stored in a new array [4, 8, 12] |
|  |  |  |
|  |  |  |