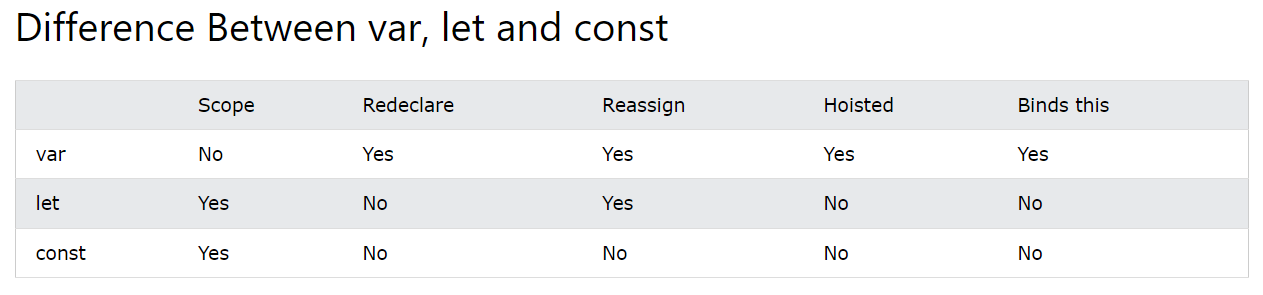
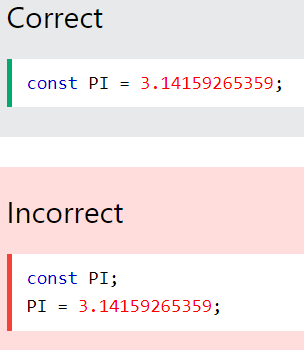
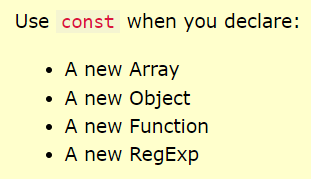
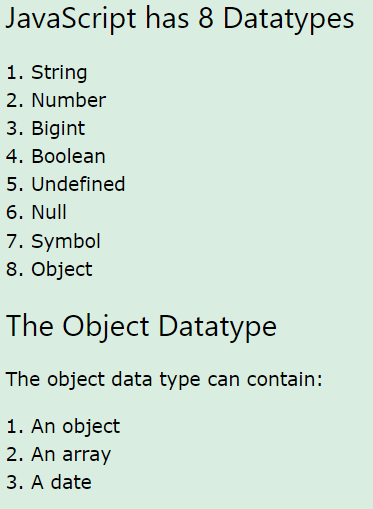
* Javascript accepts both single and double quotes.

|  |  |
| --- | --- |
| * Vars can be redeclared   Var name = “Rajesh”;  Var name; // This will work and name will still be “Rajesh” | * Let or const can’t be redeclared   let name = “Rajesh”;  let name; // This will not work as let is a block scope variable |
| * Var has global scope or can have function scope | * Let has block scope , let is introduced as part of ES6 (since 2015) |
| * Variables declared with Var can be redeclared   Ex : var x = 10;  var x; // x value will be 10 | * Variables declared with let can’t be redeclared.   Ex : let x = 10;  let x; // errors |
|  |  |



* const value must be assigned when declared.



* 
* ‘const’ doesn’t define a constant value, it defines constant reference to a value. We can update the elements of an array, properties of constant object.
* Datatypes: 

**Const:**

* It can be used to define: Arrays, Objects, Functions & RegExp. It doesn’t define a constant value but it defines a constant reference to a value.
* We can’t reassign a const array, object or value but we can change the elements of const array and change properties of object.



* Variables declared with const are also hoisted to the top but not initialized.

**Objects**:

* Objects can declared using ‘new’ keyword

Ex : const somevalue = new Object();

somevalue.firstName = “Rajesh”;

somevalue.lastName = “Vemulakonda”;

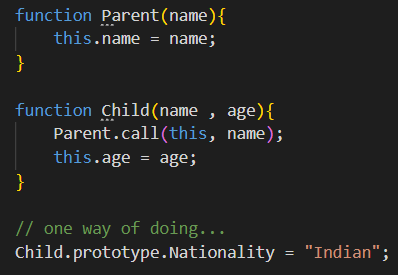
* Objects can be declared using {} syntax

Ex: const somevalue = {};

somevalue.firstName = “Rajesh”;

somevalue.lastName = “Vemulakonda”;

* **Prototype-based-inheritance 🡪** For function constructors we have prototype based inheritance. In JavaScript, every object is internally linked to it’s prototype. When we try to access a property or method and if it’s not found in it’s definition then JavaScript automatically looks up the prototype chain to find it.

****

**Memory Efficiency**: If we add a new property or method inside a function constructor that it will create copy for every object. If we use prototype all instances share one property or method.

**Inheritance**: JavaScript uses prototype based inheritance.

**Extensibility**: Without modifying existing code we are extending new methods.

In Javascript, Arrays use numbered indexes and objects use named indexes.

How to find if an object is an Array ?

Solution 1: Use Array.isArray(<ArrayName>) ex: Array.isArray(cars) // where cars is an array

Solution 2: (cars instanceof Array)

**Array Methods**:

|  |  |
| --- | --- |
| **Method** | **Usage** |
| .Length | This property returns the length of array |
| .toString() | Converts the array to a string |
| .join() | It is also like toString()  but joins the array into a string with an operator  const fruits = ["Banana", "Orange", "Apple", "Mango"];  fruits.join(" \* "); // Banana \* Orange \* Apple \* Mango |
| at() | Gets the element at an index Ex : const cars = ['Honda','BMW','Audi'];  cars.at(2) // returns 'Audi' |
| push() | adds element to the last of array |
| pop() | removes element to the last of array |
| shift() | removes the first element and shifts all the elements to lower index |
| unshift() | Adds a new element at the beginning of the array  const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.unshift("Lemon"); // Lemon,Banana,Orange,Apple,Mango |
| delete() | const fruits = ["Banana", "Orange", "Apple", "Mango"]; delete fruits[0]; // deletes Banana but leaves undefined holes in the array |
| concat() | concat() means to join so in case of arrays they mean to join arrays end to end   const myGirls = ["Cecilie", "Lone"]; const myBoys = ["Emil", "Tobias", "Linus"]; const myChildren = myGirls.concat(myBoys); // Cecilie,Lone,Emil,Tobias,Linus |
| copyWithin() | copies array elements to another position in the array  const fruits = ["Banana", "Orange", "Apple", "Mango"]; fruits.copyWithin(2,0); // Banana,Orange,Banana,Orange |
| flat() | It is used to reduce the dimensionality of an array, it is used to convert multi-dimensional array into single array.  const myArr = [[1,2],[3,4],[5,6]]; const newArr = myArr.flat(); // 1,2,3,4,5,6 |
| flatMap() | It first maps all the elements of the array and then creates a new array by flattening the array.  const myArr = [1, 2, 3, 4, 5,6]; const newArr = myArr.flatMap(x => [x, x \* 10]); // 1,10,2,20,3,30,4,40,5,50,6,60 |
| splice() | This is used to add new items to an array. The first parameter(2) signifies the position where new items have to added and second parameter (2) signifies the number of elements to be removed.  const fruits = ["Banana", "Orange", "Apple", "Mango"];   let removed = fruits.splice(2, 2, "Lemon", "Kiwi");  fruits = ["Banana","Orange","Lemon","Kiwi"]; removed = ["Apple","Mango"]; |
| slice() | slices out a piece of array into a new array.  const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"]; const citrus = fruits.slice(1); // Orange,Lemon,Apple,Mango |
| indexOf() | Searches array elements and returns it's index position |
| lastIndexOf() | returns the last occurrence of the specified element |
| includes() | This method checks if an element is present in an array |
| find() | The find() method returns the value of the first array element that passes a test function.  const numbers = [4, 9, 16, 25, 29]; let first = numbers.find(myFunction); // 25  function myFunction(value, index, array) {  return value > 18; } |
| findIndex() | returns the index of the first array element that passes a test function. const numbers = [4, 9, 16, 25, 29]; let index = numbers.findIndex(myFunction); // 3  function myFunction(value, index, array) {  return value > 18; } |
| findLast() | It will start from the end of an array and returns the value of the first element that satisfies the condition.   const temp = [27, 28, 30, 40, 42, 35, 30]; let high = temp.findLast(x => x > 40); // 42 |
| sort() | sorts an array  const fruits = [4, 3, 2 ,1]; fruits.sort(); // [1,2,3,4]; |
| toSorted() | sorts and gives a new array without altering the original array |