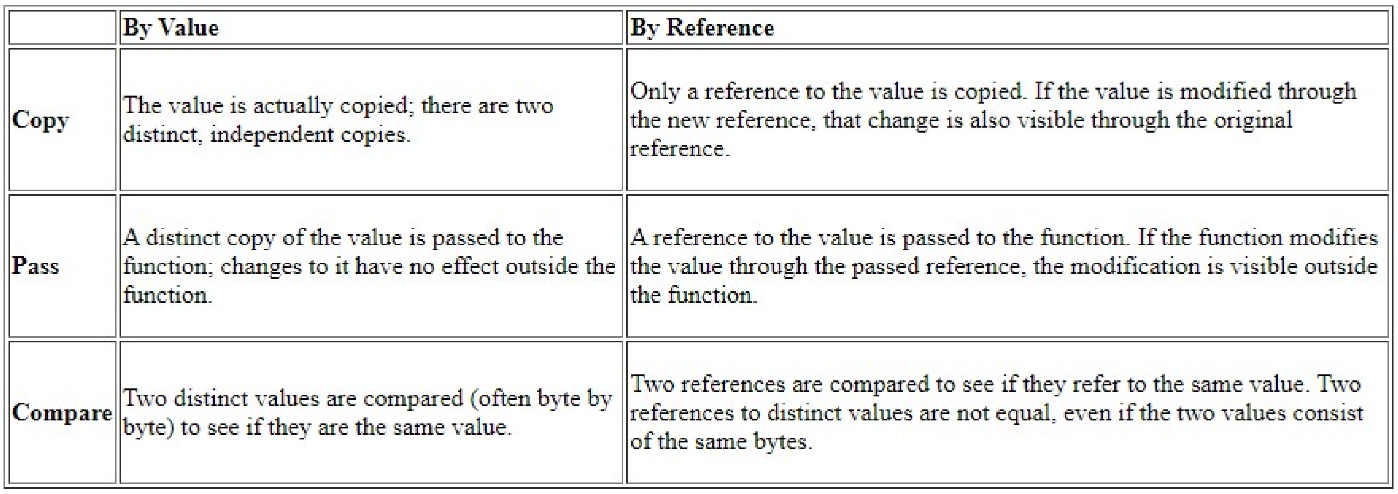
Difference between copy by value and copy by reference

When you assign a variable, it is a reference to an object not the object itself. When you copy an object b = a both variables will point to the same address.

This behaviour is called **copy by reference value**.

Strictly speaking in Ruby and JavaScript everything is copied by value.

When it comes to objects though, the values happen to be the memory addresses of those objects. Thanks to this we can modify values that sit in those memory addresses. Again, this is called copy by reference value but most people refer to this as copy by reference.



	By Value	**By Reference**
Copy	The value is actually copied; there are two distinct, independent copies.	Only a reference to the value is copied. If the value is modified through the new reference, that change is also visible through the original reference.
Pass	A distinct copy of the value is passed to the function; changes to it have no effect outside the function.	A reference to the value is passed to the function. If the function modifies the value through the passed reference, the modification is visible outside the function.
Compare	Two distinct values are compared (often byte by byte) to see if they are the same value.	Two references are compared to see if they refer to the same value. Two references to distinct values are not equal, even if the two values consist of the same bytes.

How to copy by value a composite datatype

In JavaScript, composite datatypes (arrays, objects etc.) are passed by reference to a function by default. In order to copy by value a composite datatype, we can use below methods:

1. Using Object.assign():

Using object.assign() we can copy all the data from one object to another. It is only shallow copy mechanism which means it cannot support nested objects or arrays.

Example:

```
let a1 = { name : "Ravi"}
var a2 = Object.assign({},a1);
a2.name="Teja";
console.log(a1,a2);
```

Output: {name: "Ravi"} {name: "Teja"}

In the above example, we have copied the data from a1 to a2 using Object.assign(), after that when we made the changes in a2, a1 is not affected.

2. Using spread syntax:

Using spread syntax, we can copy all the data from one object to another. It is also a shallow copy mechanism.

Example:

```
let a1 = { name : "Ravi"}
var a2 = {…a1};
a2.name="Teja";
console.log(a1,a2);
```

Output: {name: "Ravi"} {name: "Teja"}

In the above example, we have copied the data from a1 to a2 using spread syntax, after which when we made the changes in a2, a1 is not affected.

3. Using JSON.parse/stringify:

JSON.parse/stringify is a very simple one liner for deep cloning an object, but, make sure not to use Dates, functions, undefined, infinity, RegExps, Maps, Sets, Blobs, FileLists, ImageDatas, sparse Arrays, Typed Arrays or other complex types as it will not support all the data types.

Example:

```
let a1 = { name : “Ravi”}
var a2 = JSON.parse(JSON.stringify(a1));
a2.name=”Teja”;
console.log(a1,a2);
```

Output: {name: “Ravi”} {name: “Teja”}

Same example has been used to copy the data using JSON.parse/stringify.