Javascript

Comparison Operators :

Operator	Description
==	equal to
===	equal value 'and' equal type
!==	not equal value 'or' not equal type

* Adding Strings and numbers

i) Adding a number and a string will return a string

* Type Operators

i) typeof : Returns the type of variable

ii) instanceof : Return True if an object is an instance of an Object type

* Javascript evaluates expressions from left to right

e.g. let x = 4+2 + "Junaid"
=) 6Junaid

* Most programming language have many number types :

Whole numbers (integers)
byte (8-bit), short (16-bit), etc

Javascript numbers are always one type :
double (64-bit floating point)

* Two ways of creating object :

i) const person = { name : "J", age : 10 }

ii) Using new keyword :
```
const person = new Object();
person.name = "J"
person.age = "10"
```

* 'this' keyword

```
const person = { firstname : "John", lastname : "Doe", id : 5566,
                 fullname : function() { return this.firstname + " " + this.lastname; } };
```

* Javascript Objects are mutable

Objects are mutable : They are addressed by reference, not by value.
If person in an object, the following statement will not create a copy of person
```
const x = person;
```

The object x is not a copy of person. The object x is person.
The object x and the object person share the same memory address
Any changes to x will also change person.

* Deleting Properties of Object :
→ delete person.age or delete person["age"];

* Nested OBJECTS

```
my obj = { name: 'John', age = 30,
           myCars : { car1: "Ford"
                      car2 : "BMW"
                      car3 : "Fiat" } }
```

You can access nested objects using the dot notation
`myobj.myCars.car2;`

* Reduce function in java script

syntax : array.reduce (callback (accumulator, current value, current index, array), initial value)

* Accumulator : It is a accumulated value previously returned in last invocation of callback, or the initial value i
* Current value : The current element being processed in the array
* Current index : (optional) The index of current element being processed in the array.
  Starts from '0' if 'initialvalue is provided, otherwise from '1'.
* Initial value : (optional) A value to use as the first argument to the first call of the callback.
  If no initial value is provided, the first element in the array will be used as initial
  and iteration will start from second element.

Important

if supplied

accumulator value;