# Lecture-2

## → Variable Declaration :-

```
let age = 20;
age = 30;
```
① Value can be changed
② Cannot redeclared in the same scope.

```
const age = 25;
age = 27;
```
① Value cannot be reassigned after declaration.
② Must be initialized when declared.

```
var a = 10;
a = 20;
```
① Value can be changed
② Can be redeclared in the same scope.
③ var is function scoped, not block scoped.

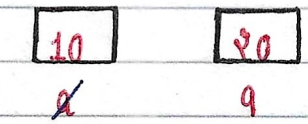
## → Data Types in JS :-

### ⊙ Primitive
1. Number (-$2^{53}$ to $2^{53}-1$)
2. String
3. Boolean
4. Undefined — variable declared but not assigned a value.
5. Null — typeof → object (bug) — No Value / Empty Value
6. Big Int — To store very large integers
7. Symbol — create unique identifiers.

### ⊙ Non-Primitive
typeof → object
1. Array
2. Object
3. function

## → Primitive data types are immutable :-

```
let a = 10;
a = 20
```




10 is not changed.
A new value 20 is assigned to a.

once a primitive value is created, it cannot be changed.

## → Non-Primitive data types are mutable :-

```
let arr = [10, 20, 30, 40]
arr[0] = 70;
```
(70 replaces 10)

The same array is modified.

The value can be changed after it is created without creating a new object.

→ Primitive data types in JS are passed by value

- When a primitive value is passed or assigned, a copy of the value is created.
- Changing one variable does not affect the other.

eg.
```
let a = 10;
let b = a;
b = 20;
console.log(a); // 10
console.log(b); // 20
```

a & b store separate copies of the value.

→ Non-Primitive data types in JS are passed by reference

- When a non-primitive value is passed, the memory reference is shared.
- Changes made through one variable affect the same object in memory.

eg.
```
let obj1 = {
    name : "Mohan",
    age : 20
}
let obj2 = obj1;
obj2.name = "Rohan";

console.log(obj1.name); // Rohan
```

Both variables points to the same object.