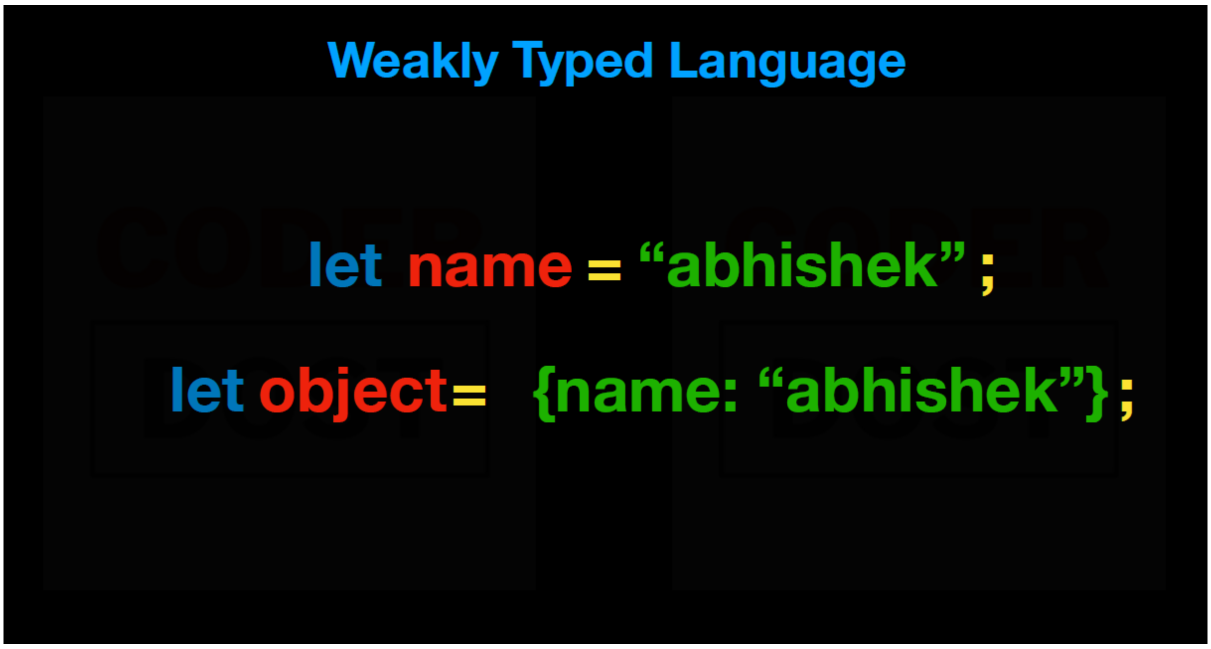
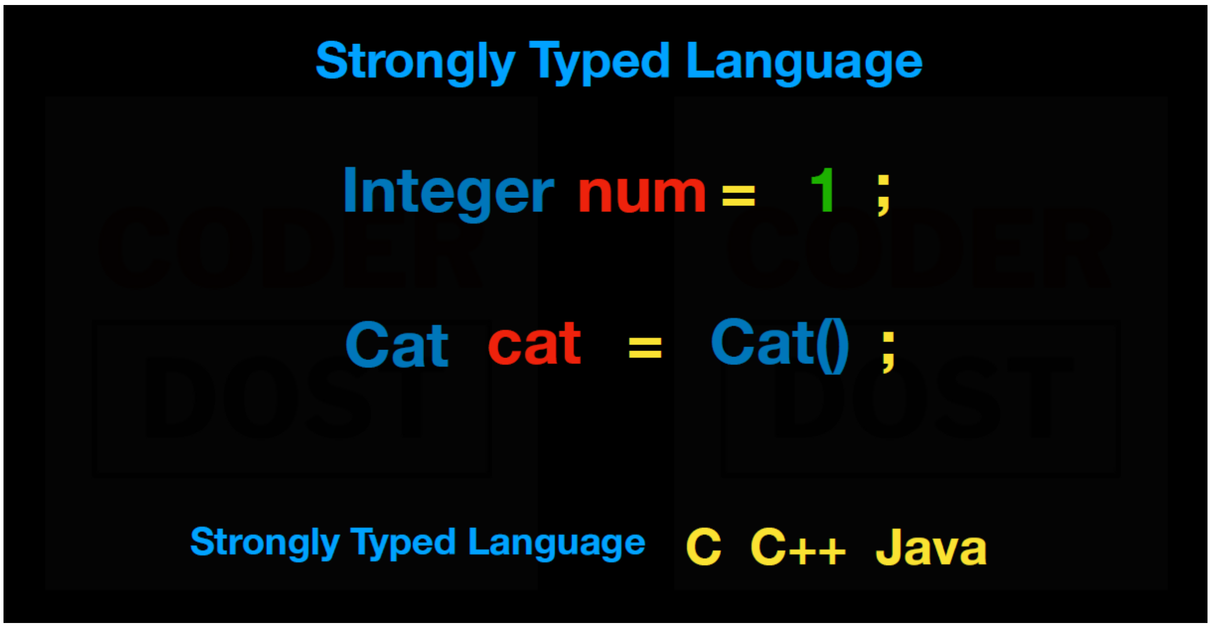
1. JavaScript Basics



Weakly Typed Language

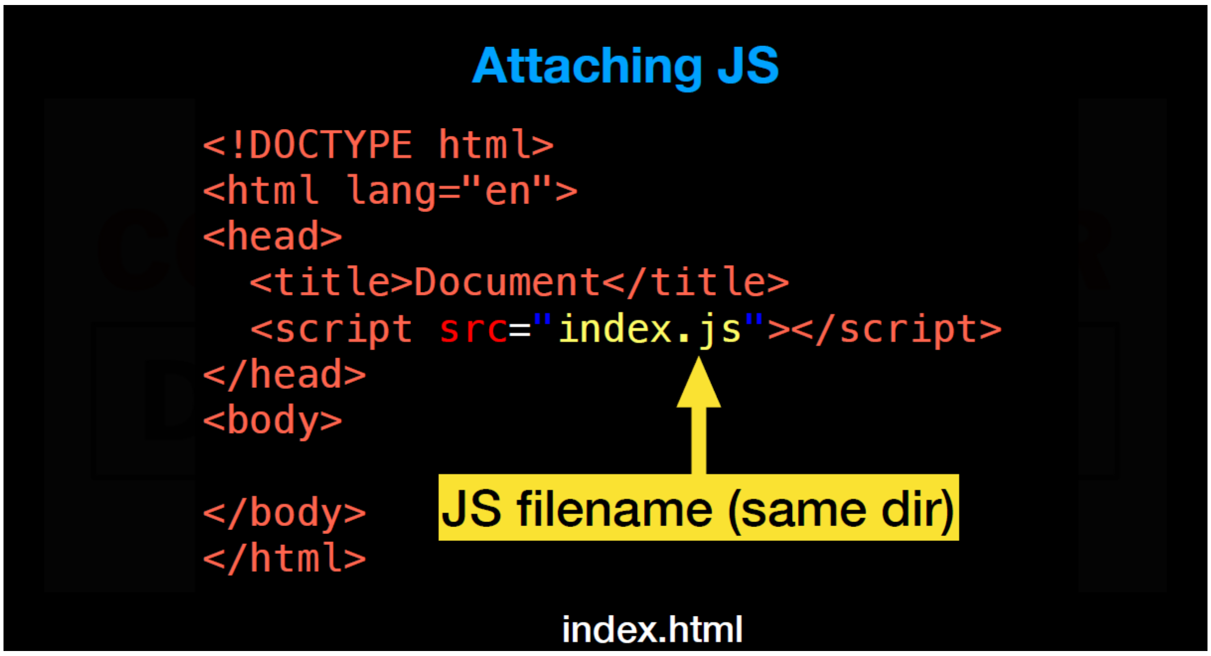
```
let name = “abhishek” ;

let object=  {name: “abhishek”} ;
```



Strongly Typed Language

```
Integer num = 1 ;

Cat cat = Cat() ;
```

Strongly Typed Language C C++ Java



Attaching JS

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
  <script src="index.js"></script>
</head>
<body>

</body>
</html>
```

JS filename (same dir)

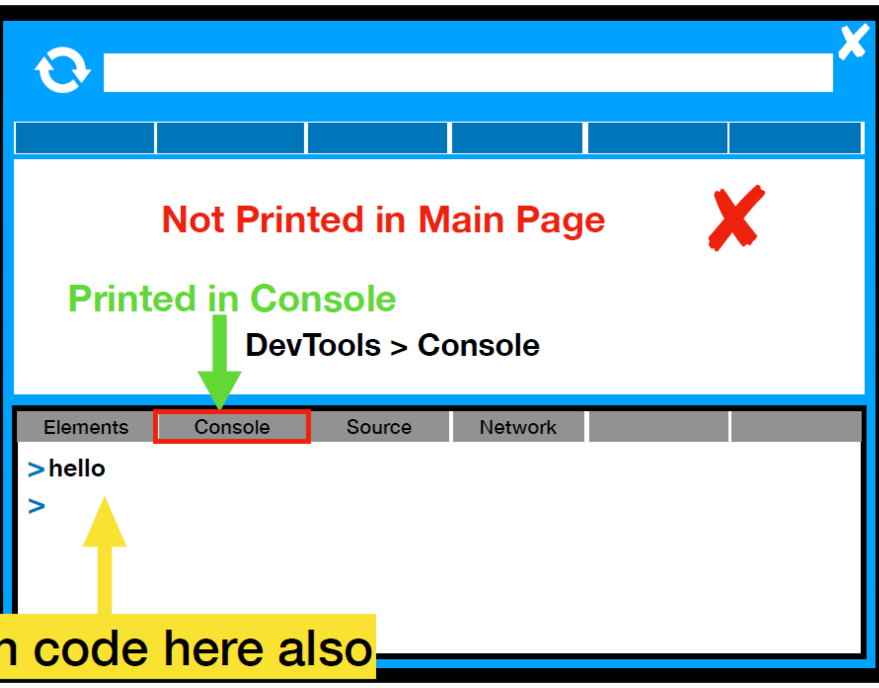
index.html

Print Statement of JavaScript

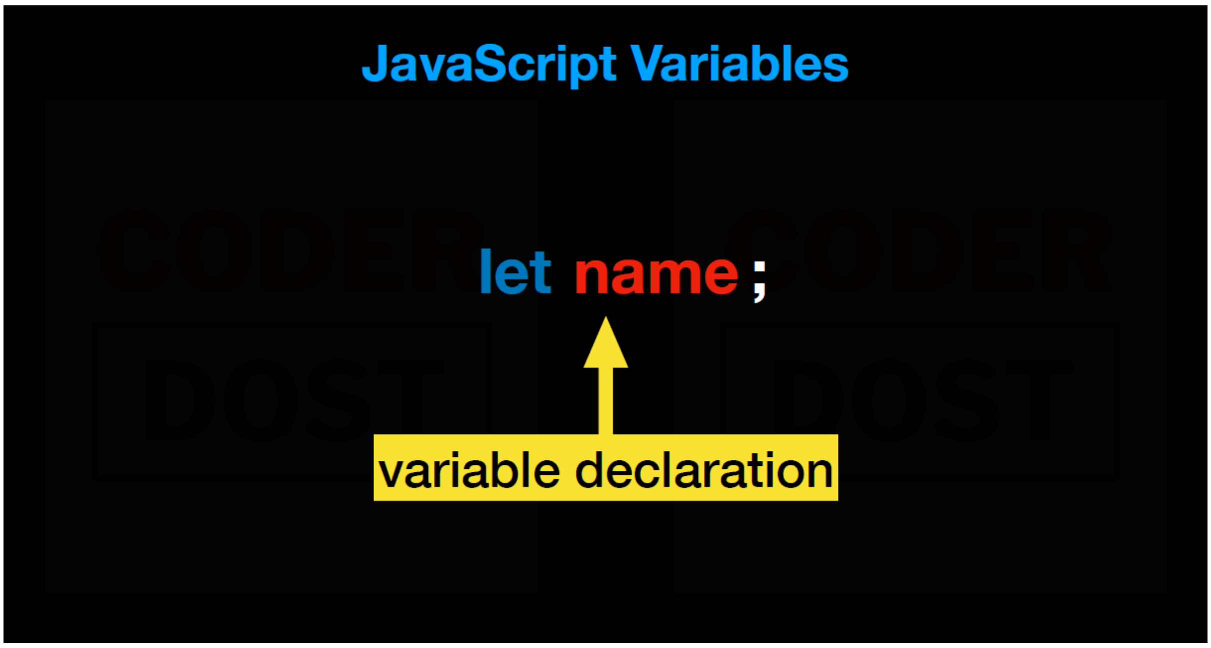
```
console.log("hello");
```

index.js

prints hello



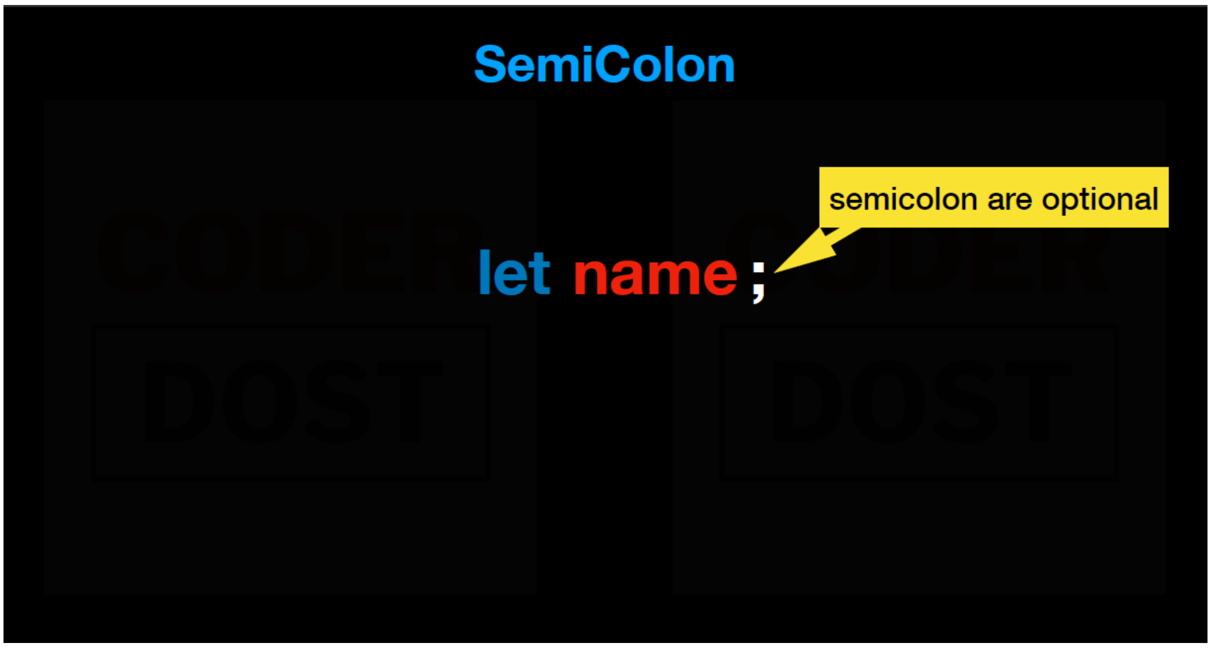
```
>hello
>
```



JavaScript Variables

```
let name;
```

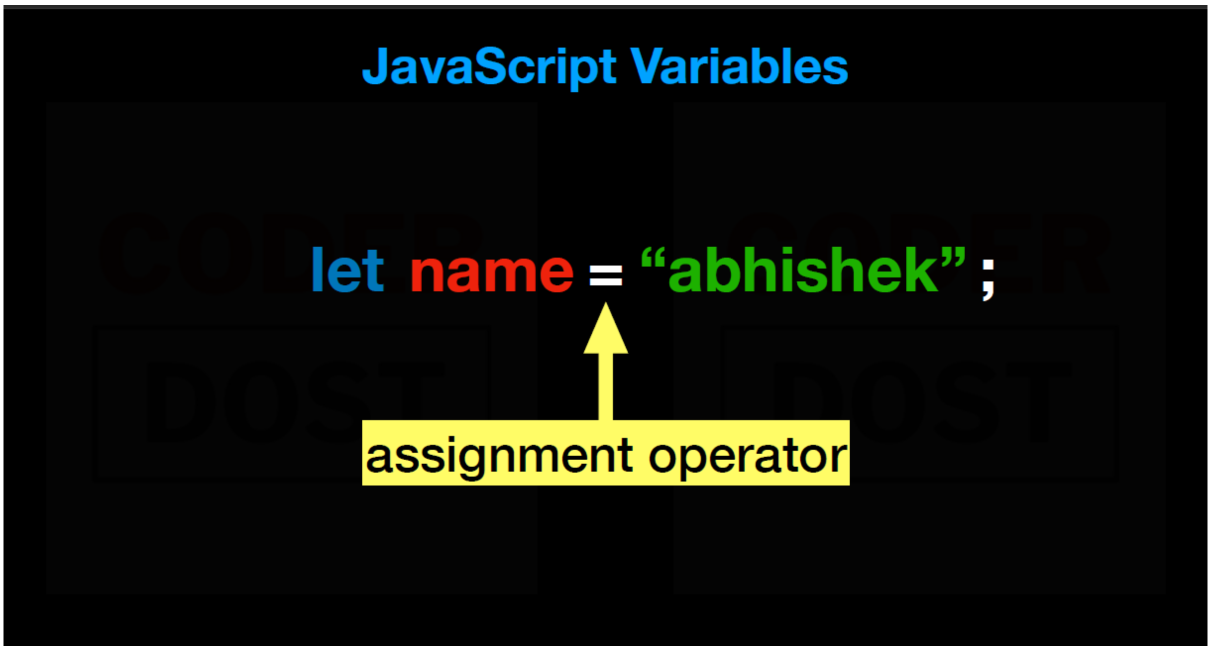
variable declaration



SemiColon

```
let name;
```

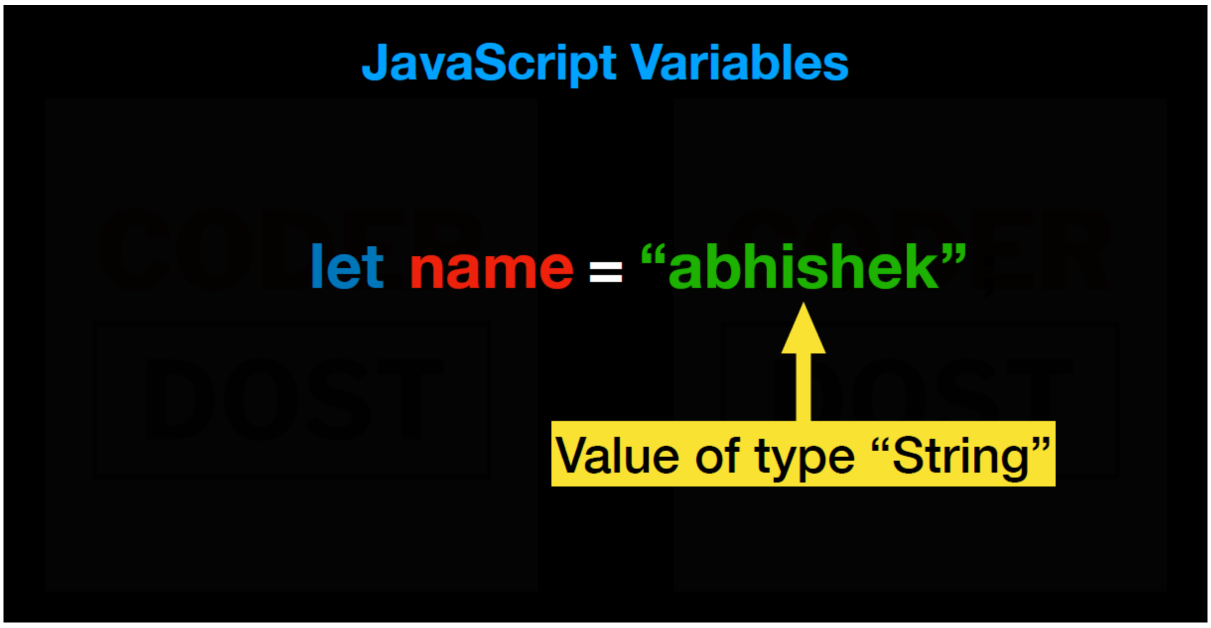
semicolon are optional



JavaScript Variables

```
let name = “abhishek” ;
```

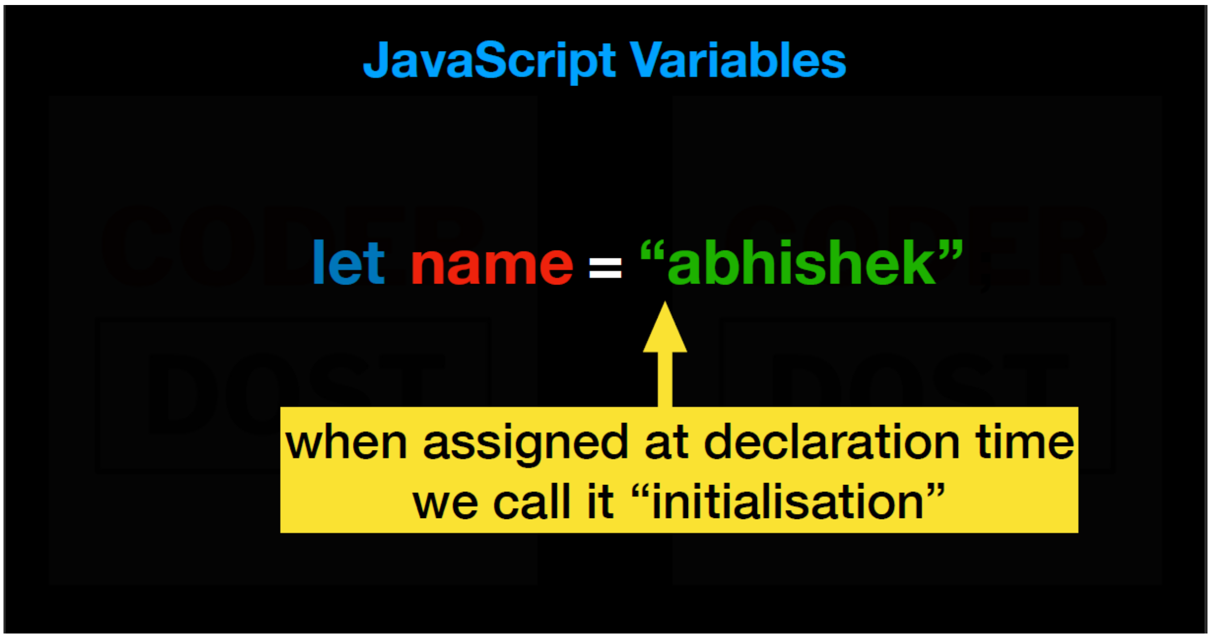
assignment operator



JavaScript Variables

```
let name = “abhishek”
```

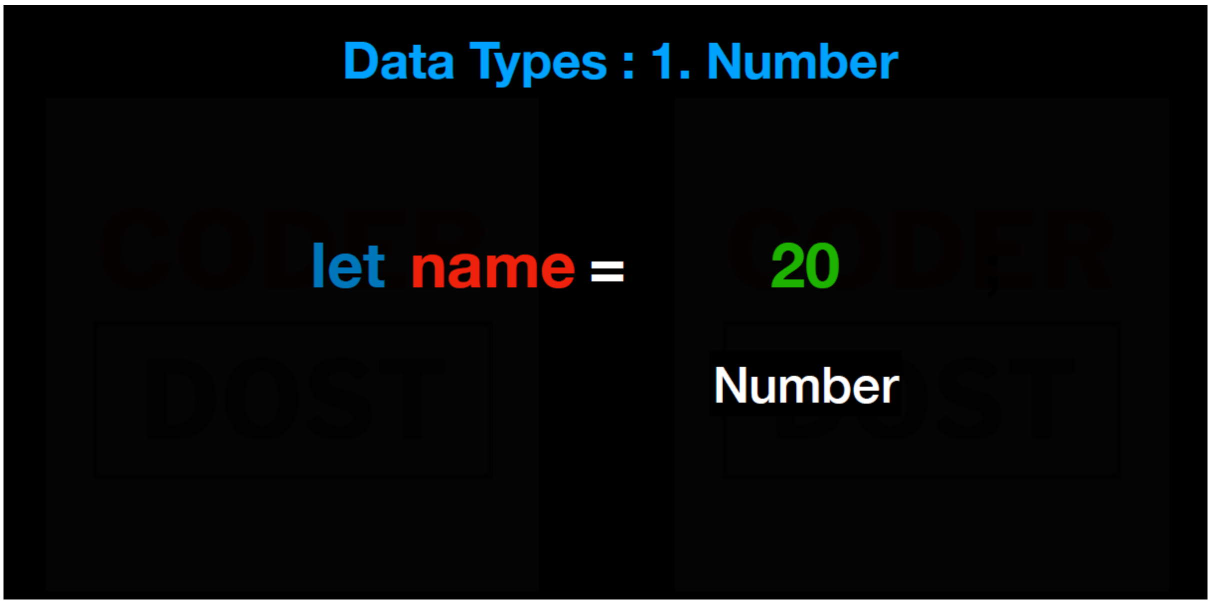
Value of type “String”



JavaScript Variables

```
let name = "abhishek"
```

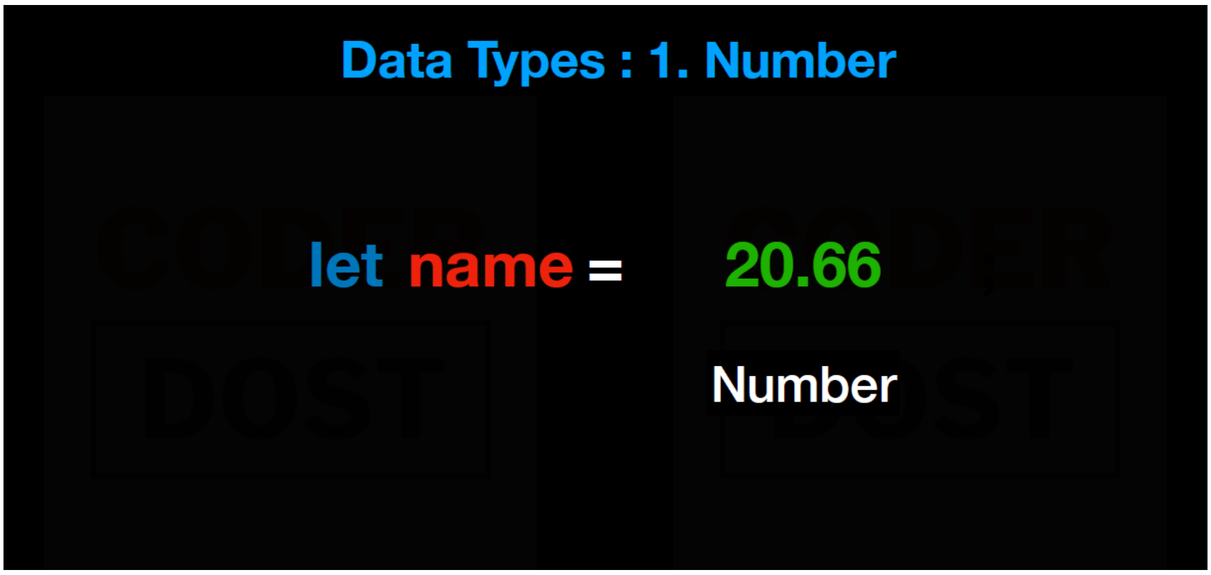
when assigned at declaration time
we call it "initialisation"



Data Types : 1. Number

```
let name = 20
```

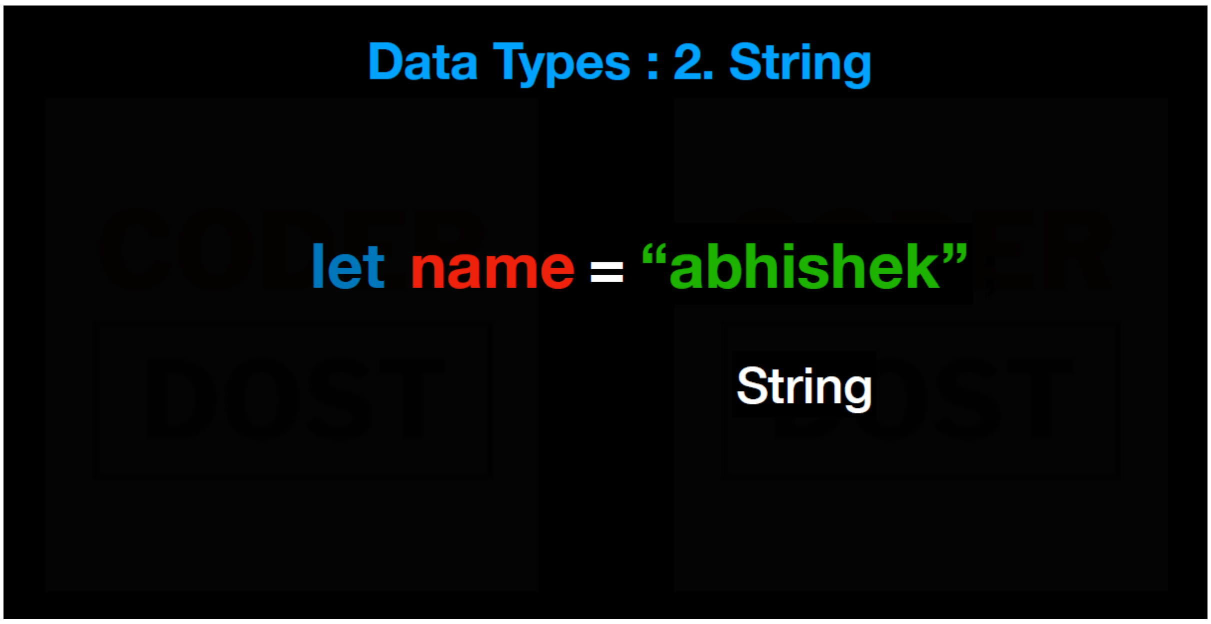
Number



Data Types : 1. Number

```
let name = 20.66
```

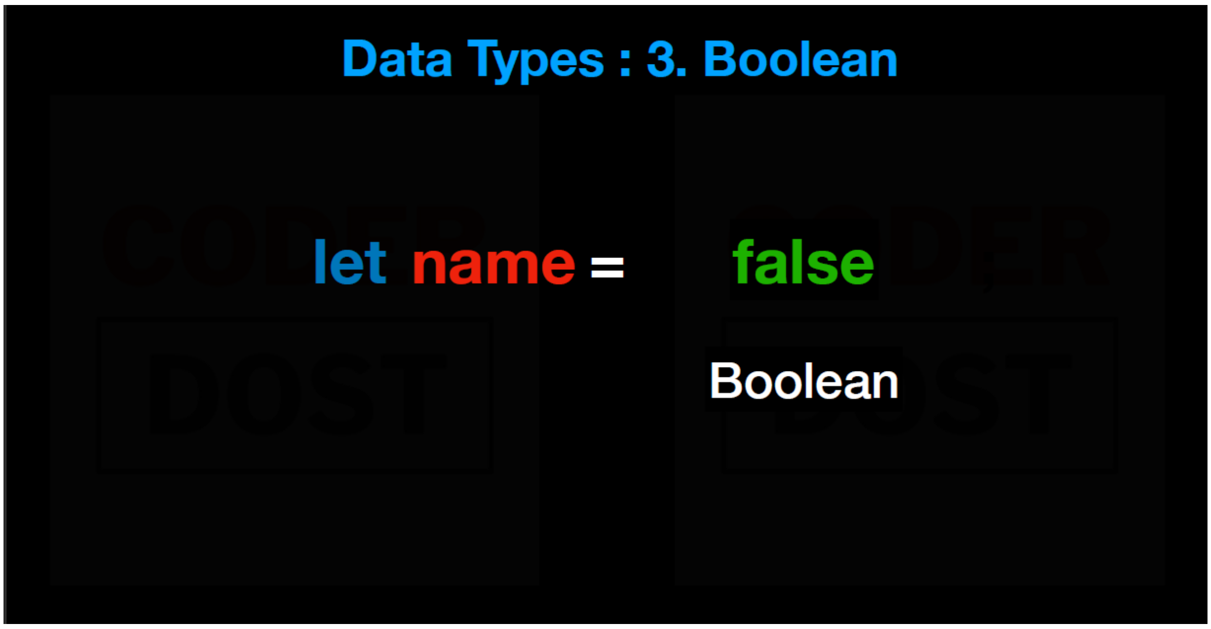
Number



Data Types : 2. String

```
let name = “abhishek”
```

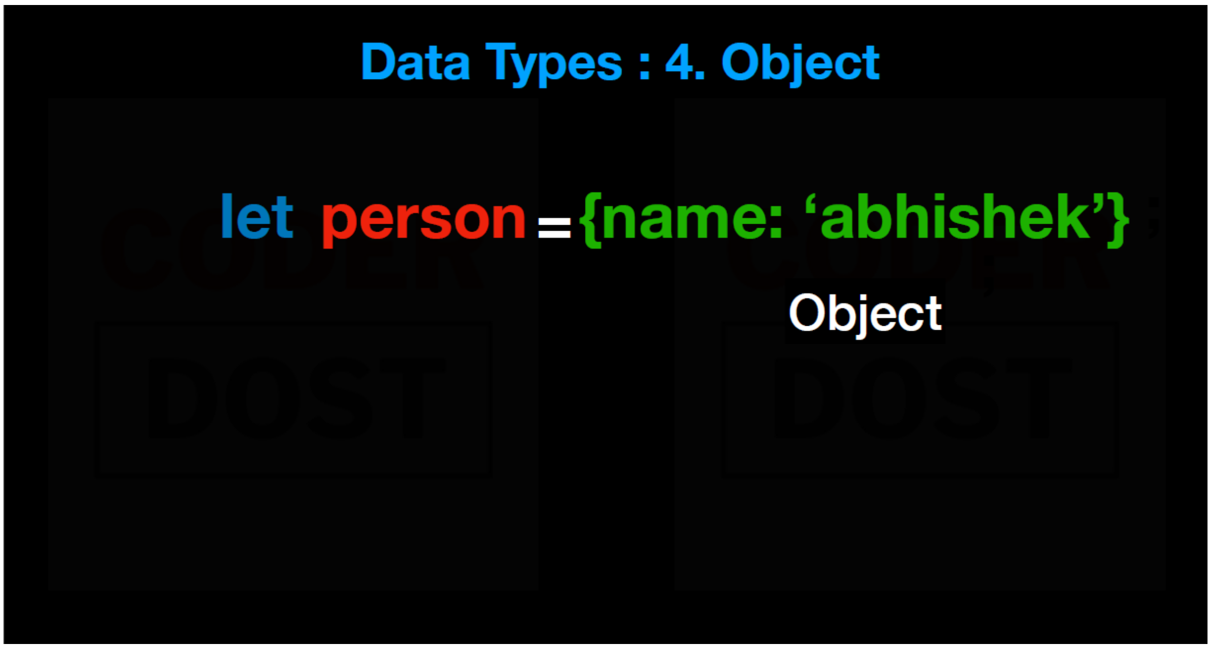
String



Data Types : 3. Boolean

```
let name = false
```

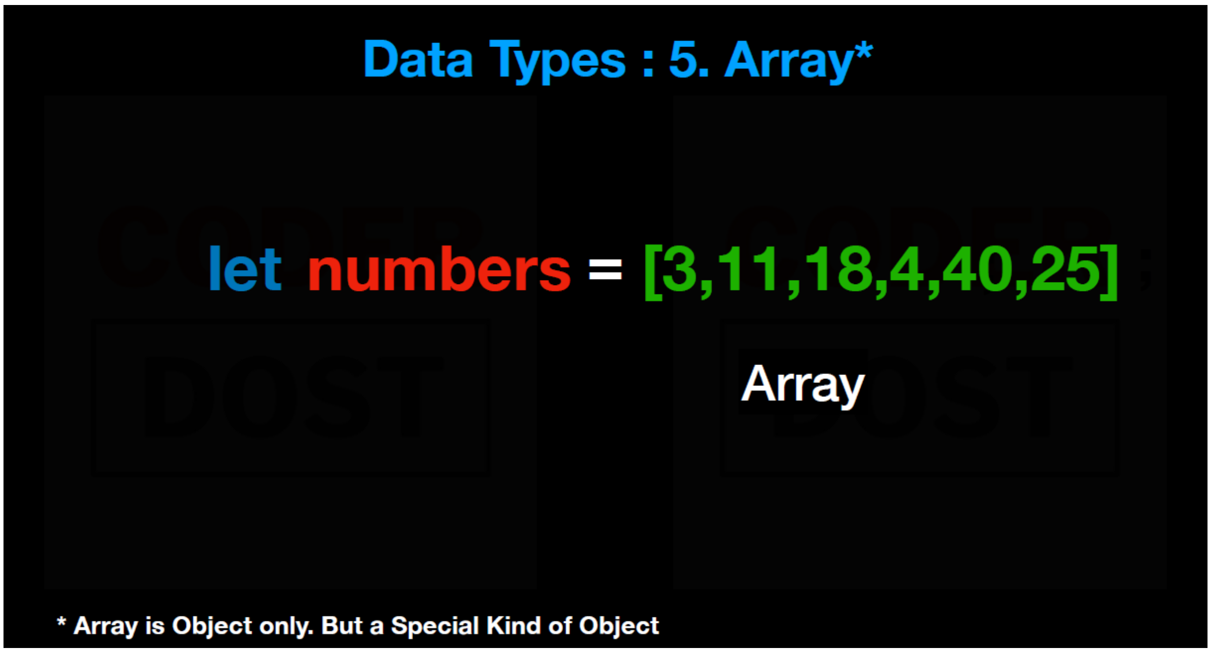
Boolean



Data Types : 4. Object

```
let person = {name: 'abhishek'}
```

Object

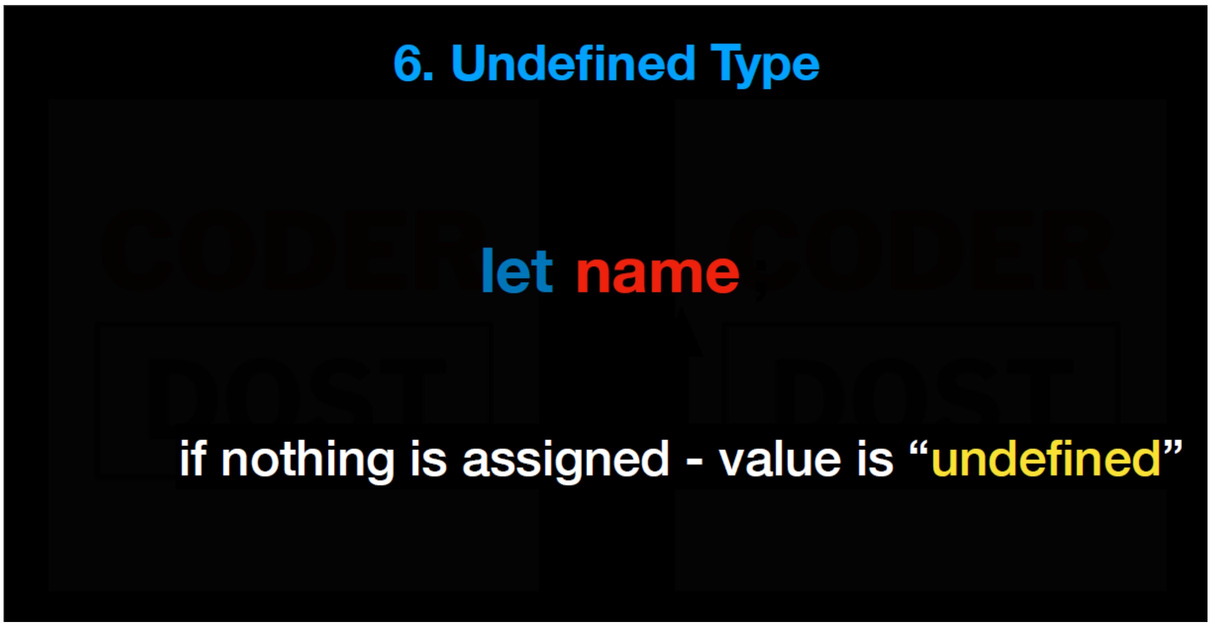


Data Types : 5. Array*

```
let numbers = [3,11,18,4,40,25];
```

Array

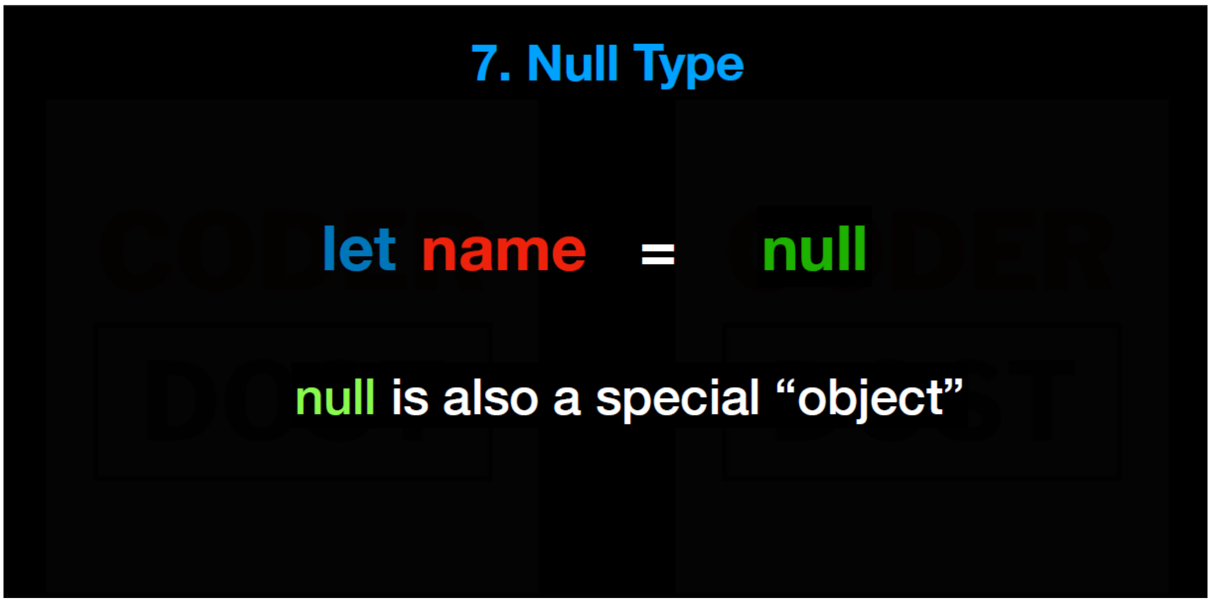
* Array is Object only. But a Special Kind of Object



6. Undefined Type

```
let name;
```

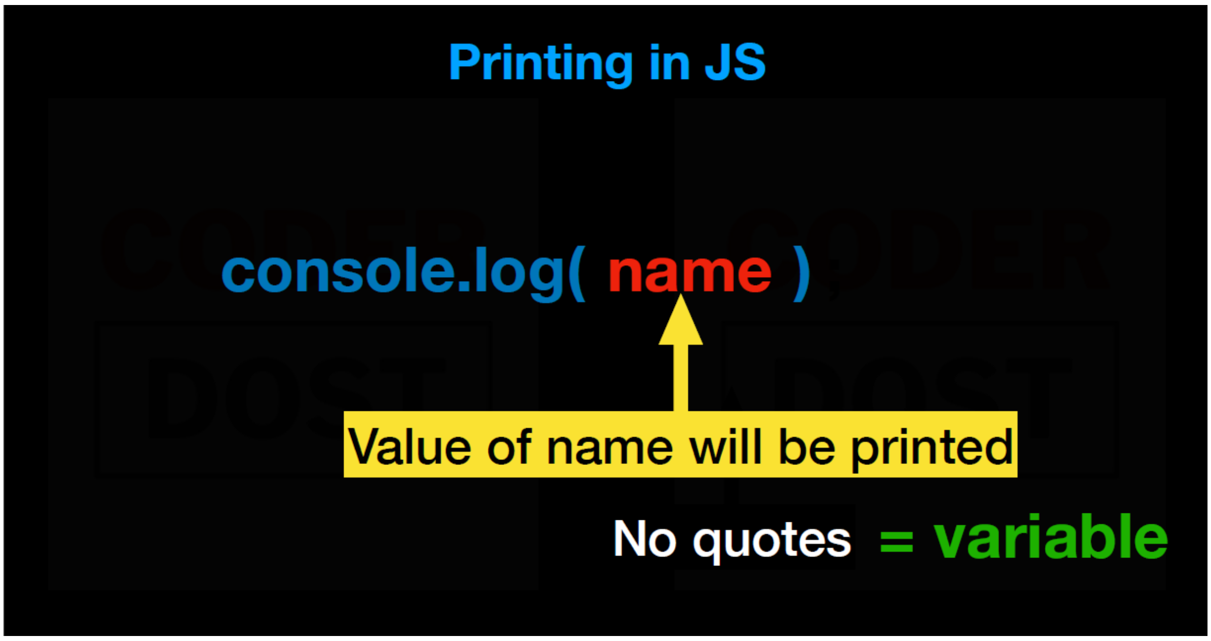
if nothing is assigned - value is “undefined”



7. Null Type

```
let name = null
```

null is also a special “object”

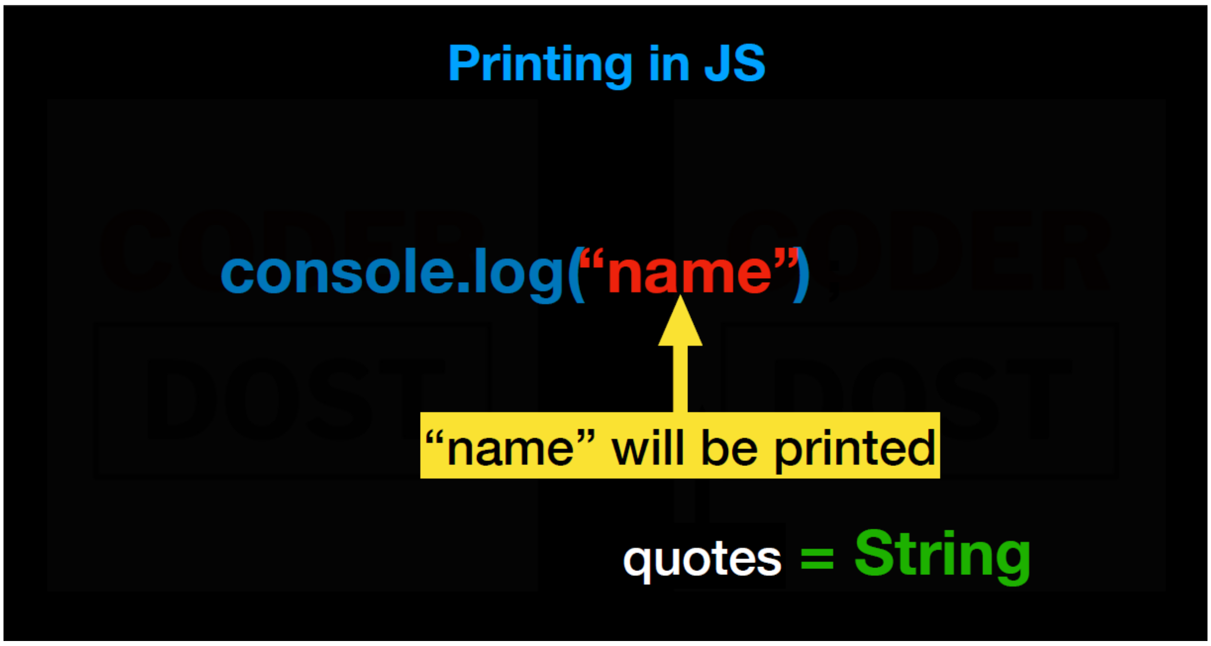


Printing in JS

```
console.log( name )
```

Value of name will be printed

No quotes = variable

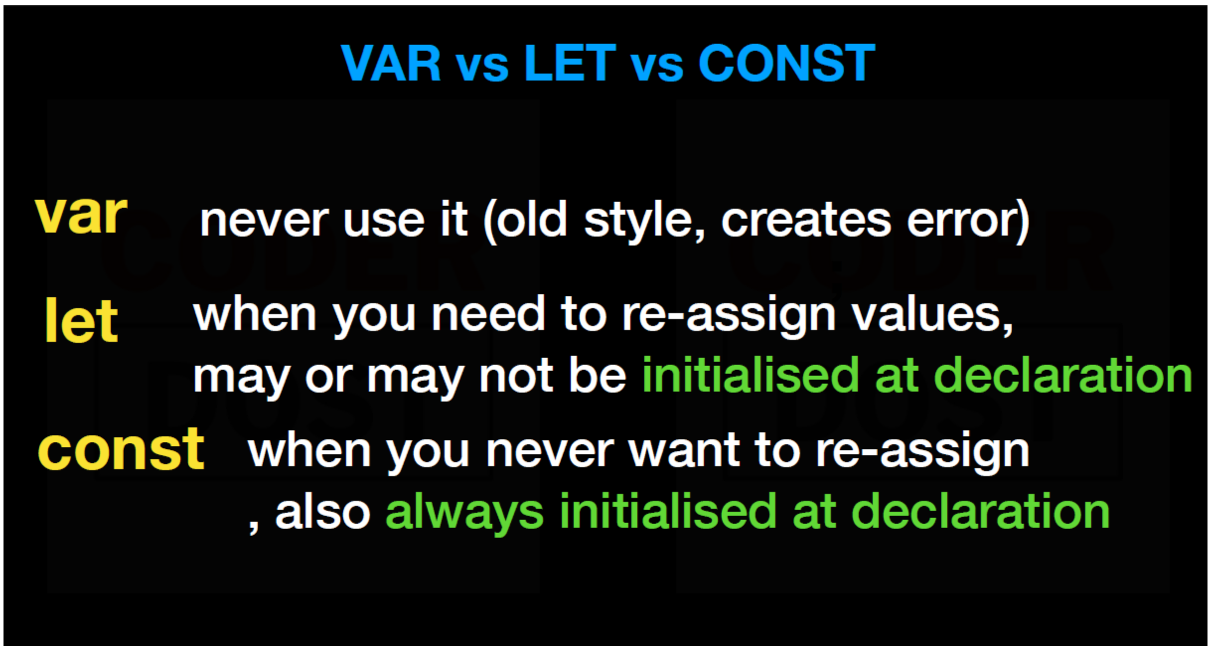


Printing in JS

```
console.log("name")
```

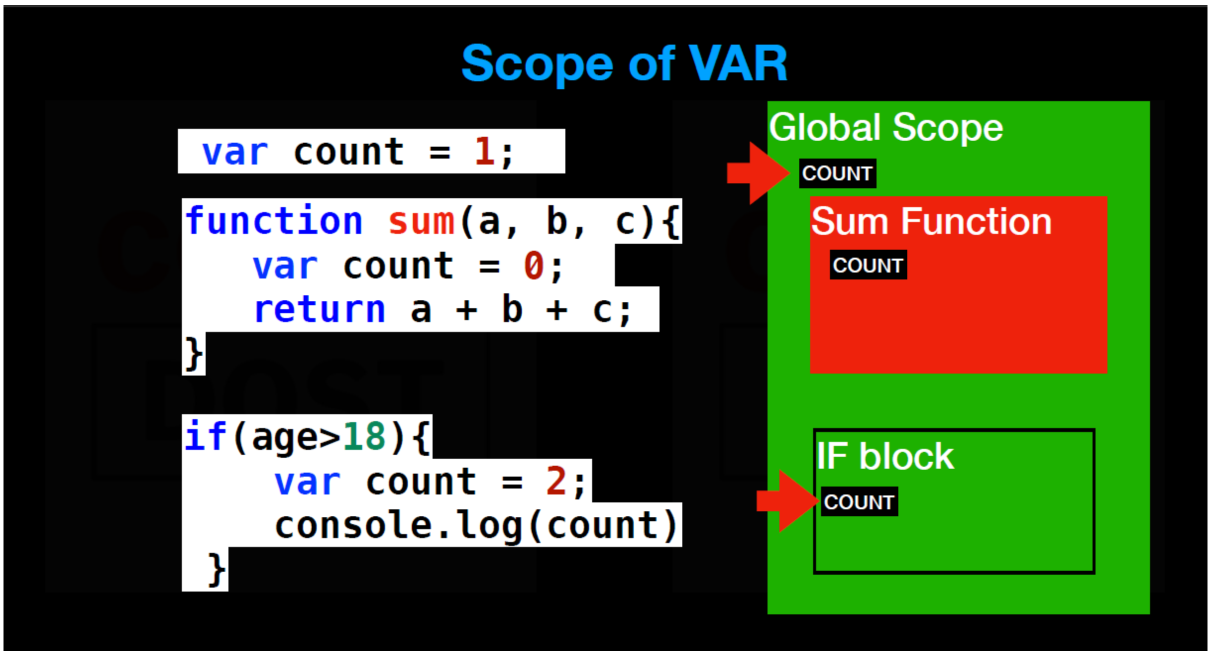
"name" will be printed

quotes = String



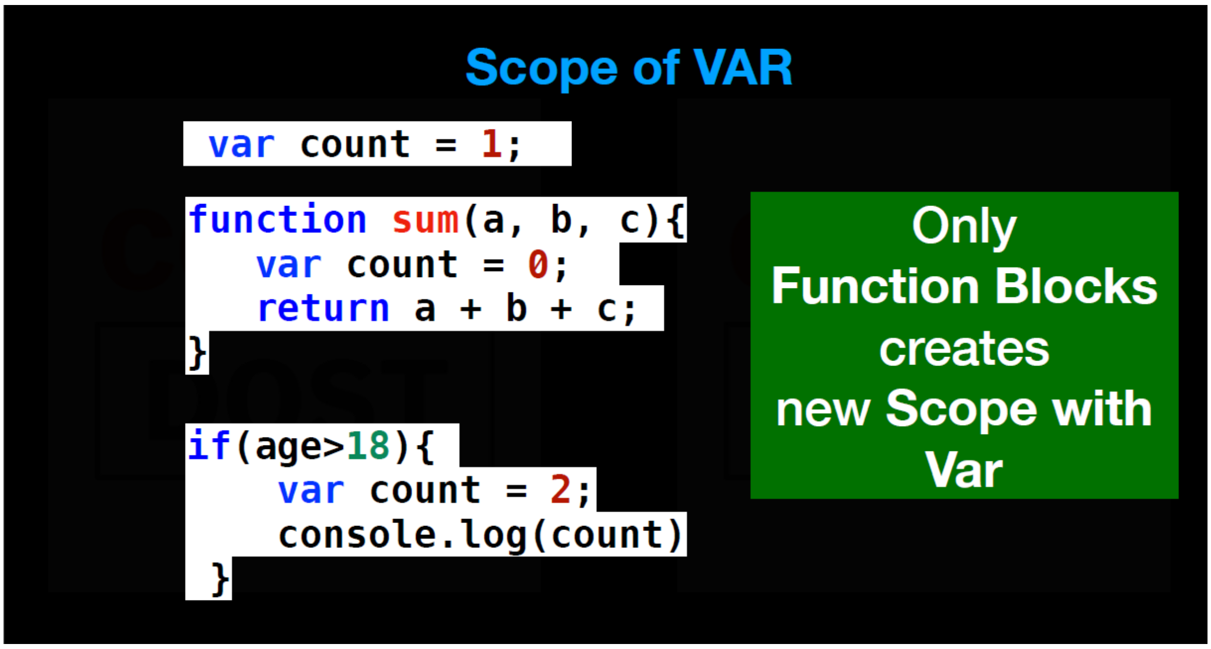
VAR vs LET vs CONST

var	never use it (old style, creates error)
let	when you need to re-assign values, may or may not be initialised at declaration
const	when you never want to re-assign , also always initialised at declaration



Scope of VAR

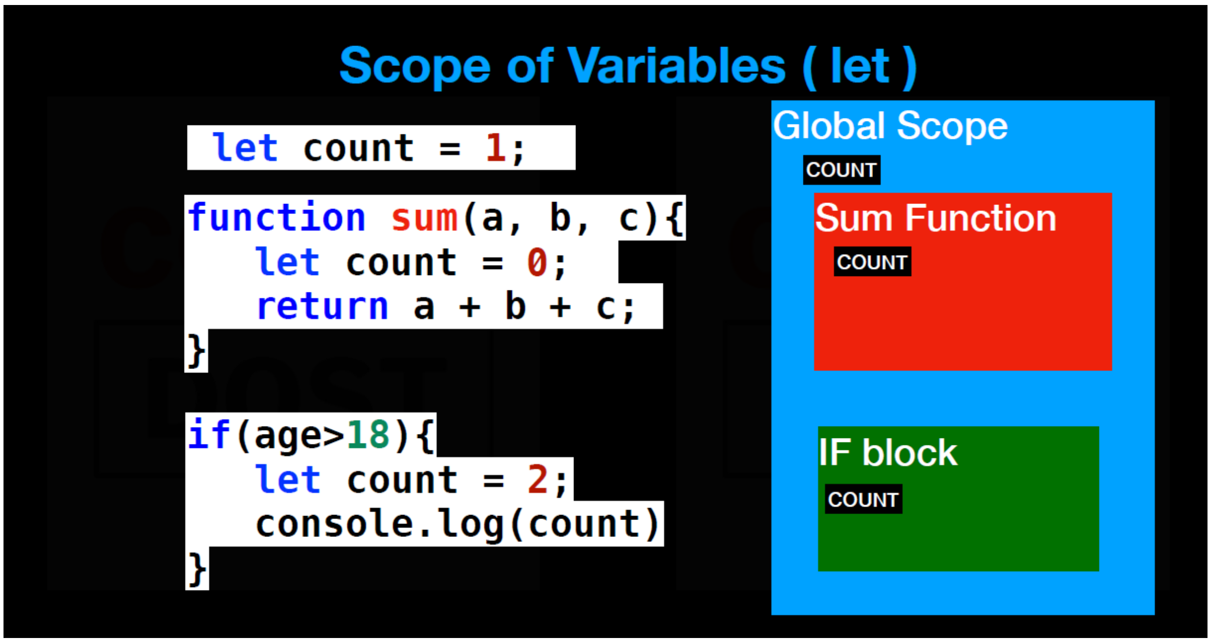
```
 var count = 1;

function sum(a, b, c){
   var count = 0;
   return a + b + c;
}

if(age>18){
     var count = 2;
     console.log(count)
 }
```



Scope of VAR

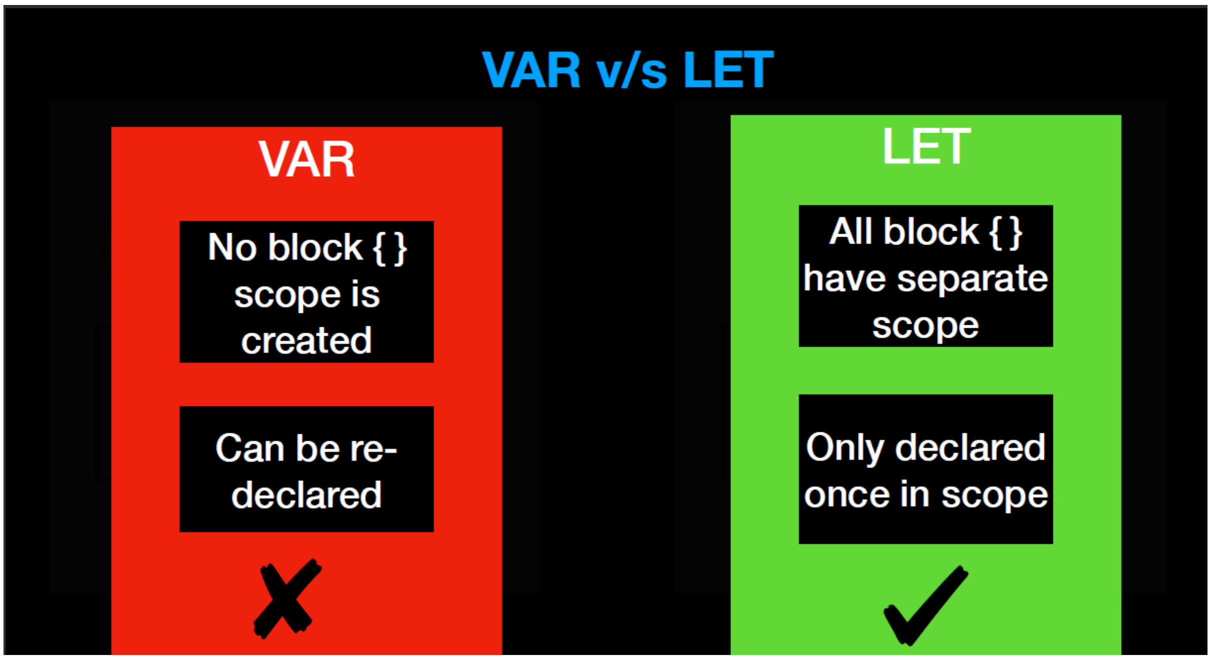
```
 var count = 1;

function sum(a, b, c){
   var count = 0;
   return a + b + c;
}

if(age>18){
     var count = 2;
     console.log(count)
 }
```

Only **Function Blocks creates new Scope with Var**



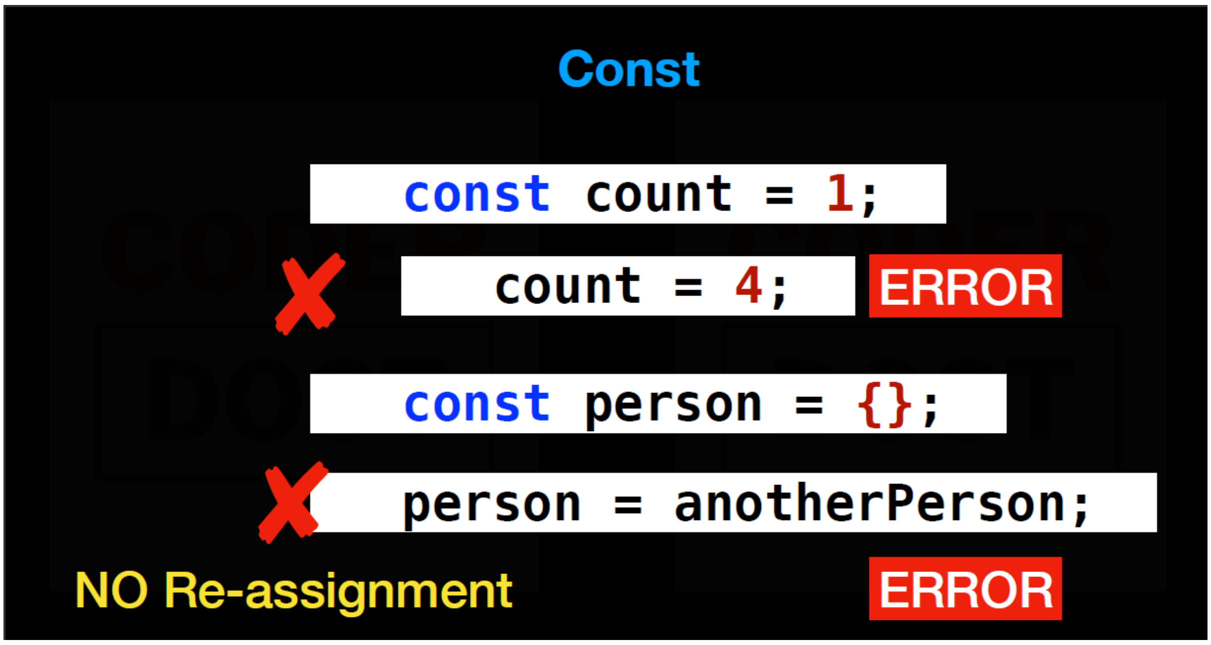
Scope of Variables (let)

```
 let count = 1;

function sum(a, b, c){
   let count = 0;
   return a + b + c;
}

if(age>18){
   let count = 2;
   console.log(count)
}
```



VAR v/s LET

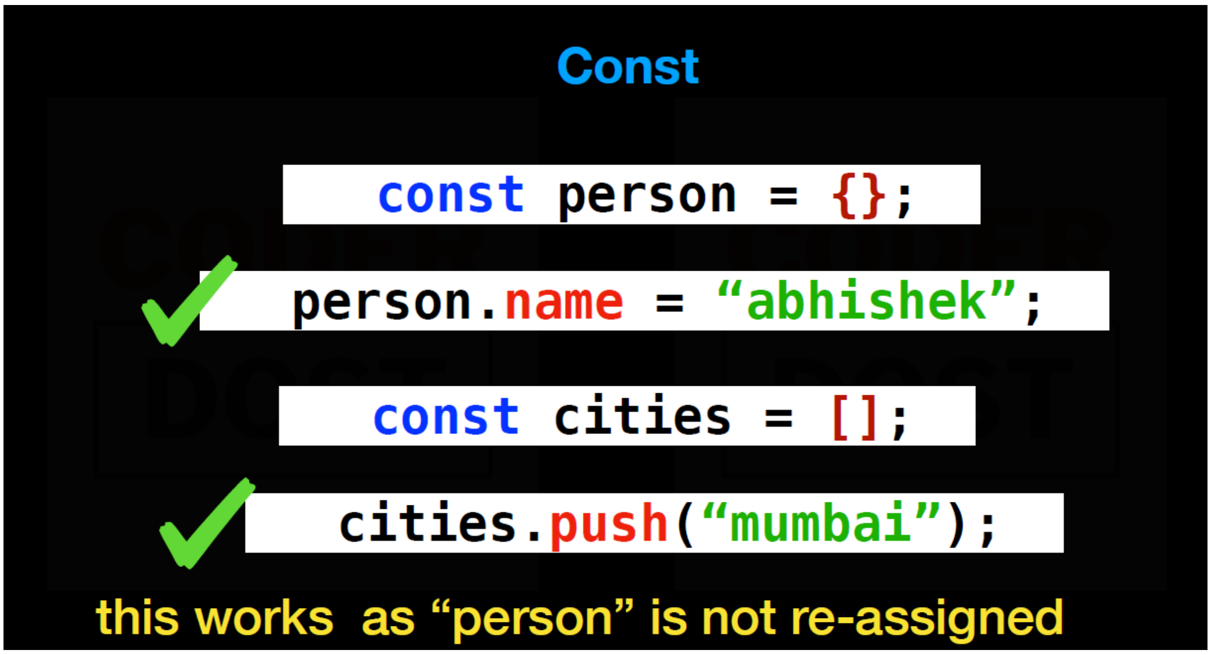
VAR	LET
No block { } scope is created	All block { } have separate scope
Can be re-declared	Only declared once in scope
✗	✓



Const

```
const count = 1;
✗ count = 4;            ERROR
const person = {};
✗ person = anotherPerson;
```

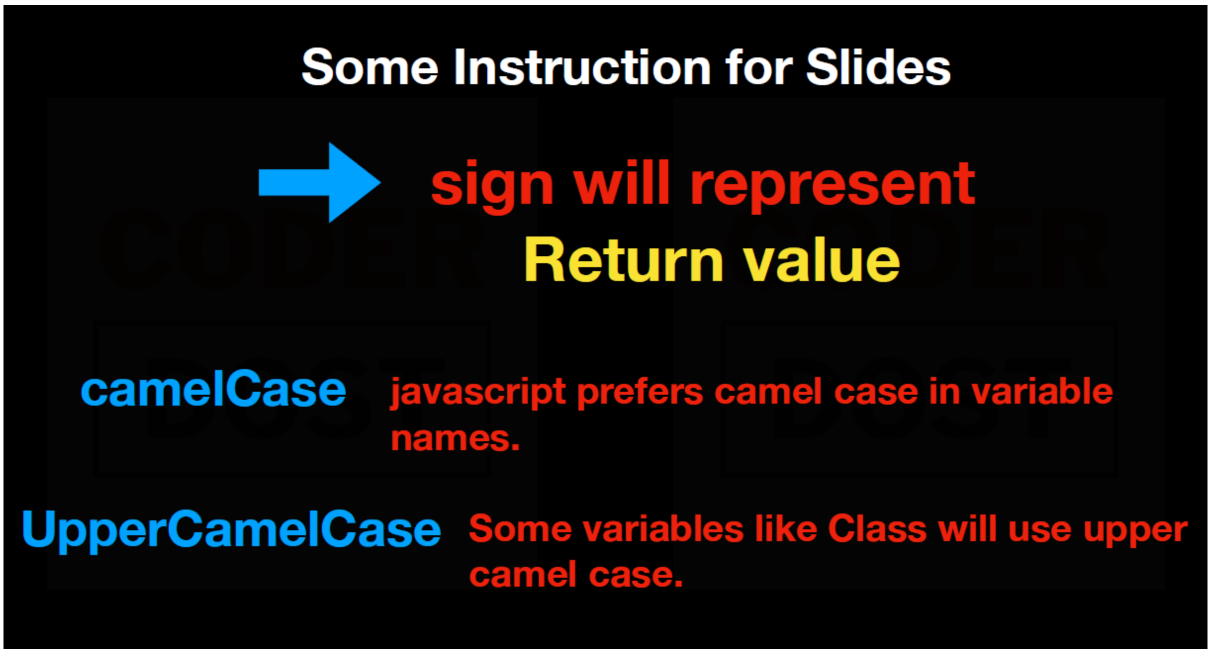
NO Re-assignment ERROR



Const

```
const person = {};
✓ person.name = “abhishek”;
const cities = [];
✓ cities.push(“mumbai”);
```

this works as “person” is not re-assigned

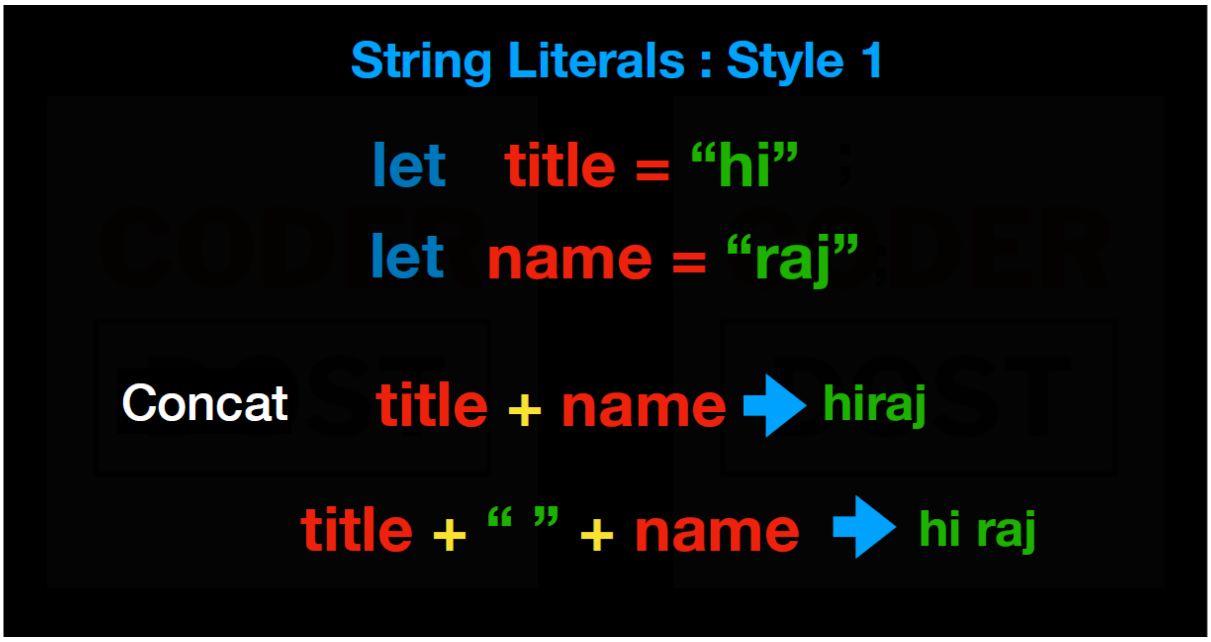


Some Instruction for Slides

→ sign will represent Return value

camelCase javascript prefers camel case in variable names.

UpperCamelCase Some variables like Class will use upper camel case.

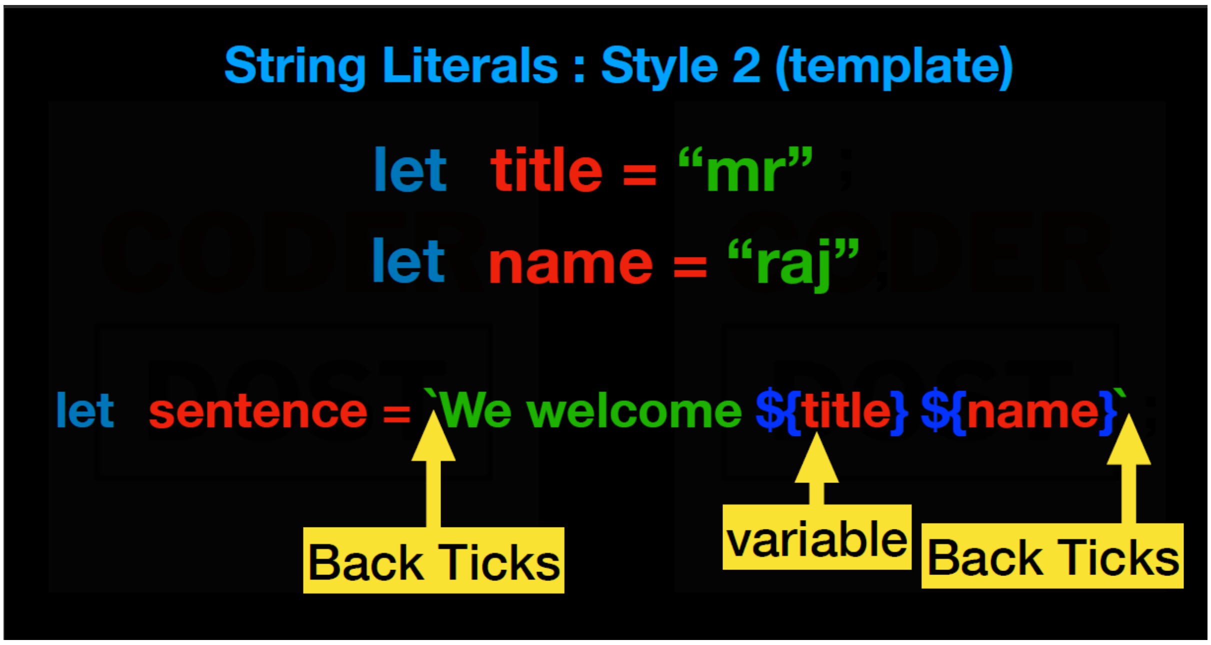


String Literals : Style 1

```
let title = “hi”;
let name = “raj”;
```

Concat title + name → hiraj

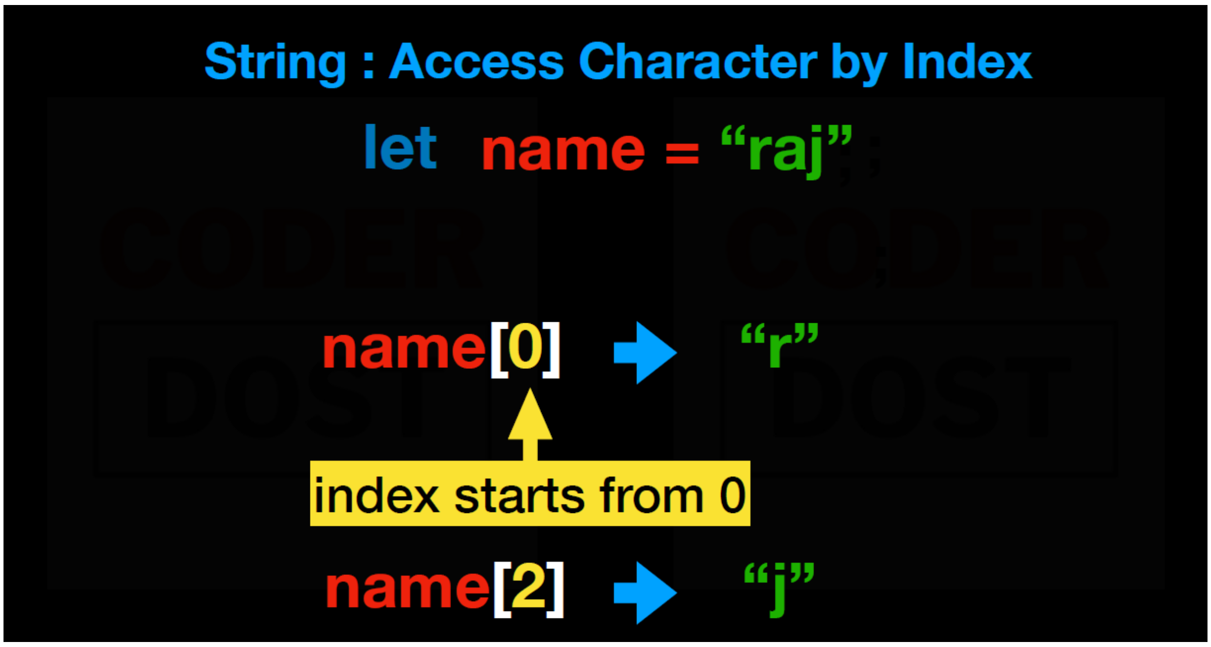
title + “ ” + name → hi raj



String Literals : Style 2 (template)

```
let  title = “mr” ;
let  name = “raj” ;

let  sentence = `We welcome ${title} ${name}` ;
```

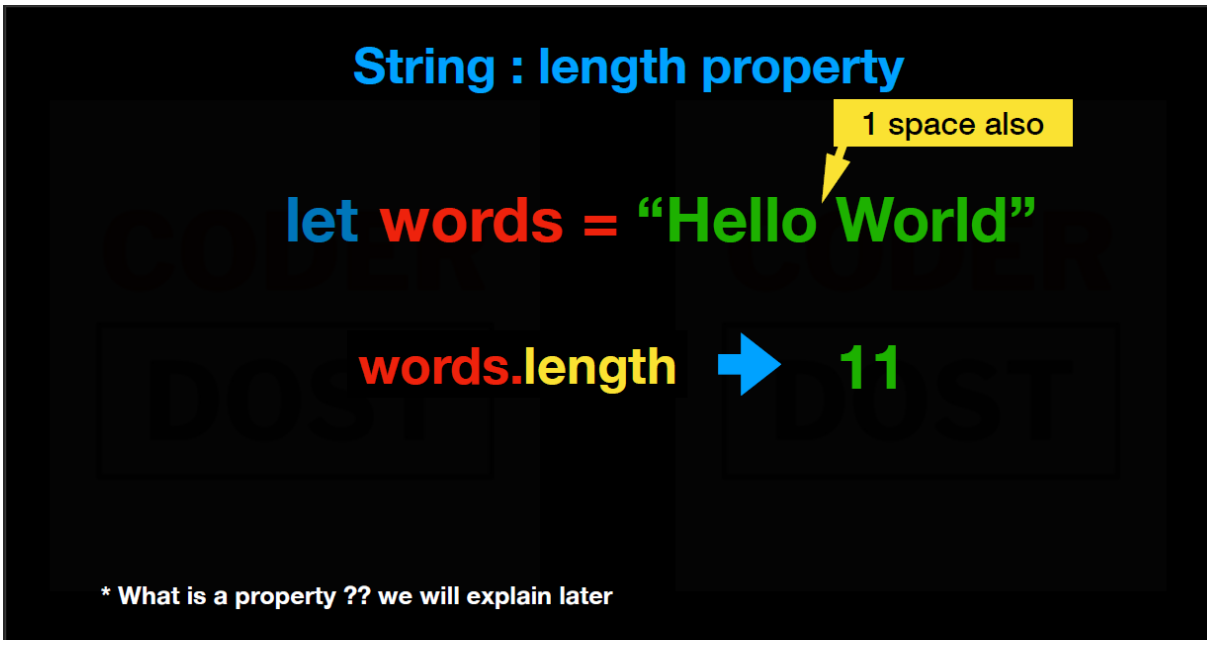
Back Ticks
variable
Back Ticks



String : Access Character by Index

```
let  name = “raj” ;

name[0]  →  “r”
name[2]  →  “j”
```

index starts from 0

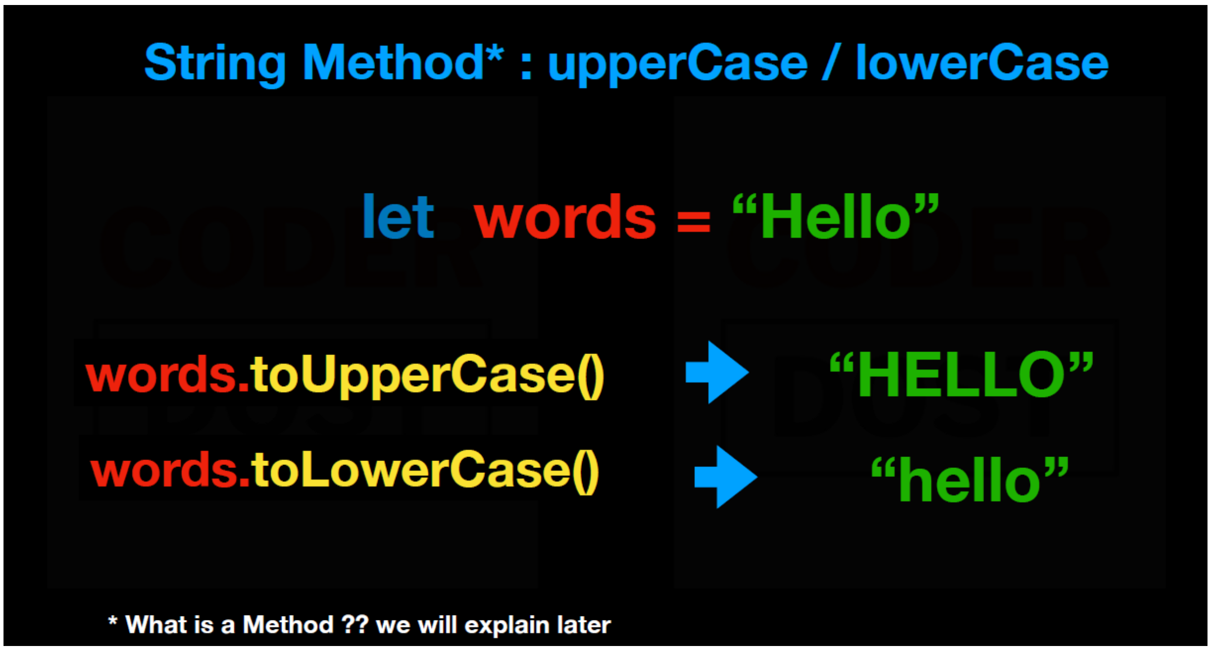


String : length property

1 space also

```
let words = “Hello World”

words.length → 11
```

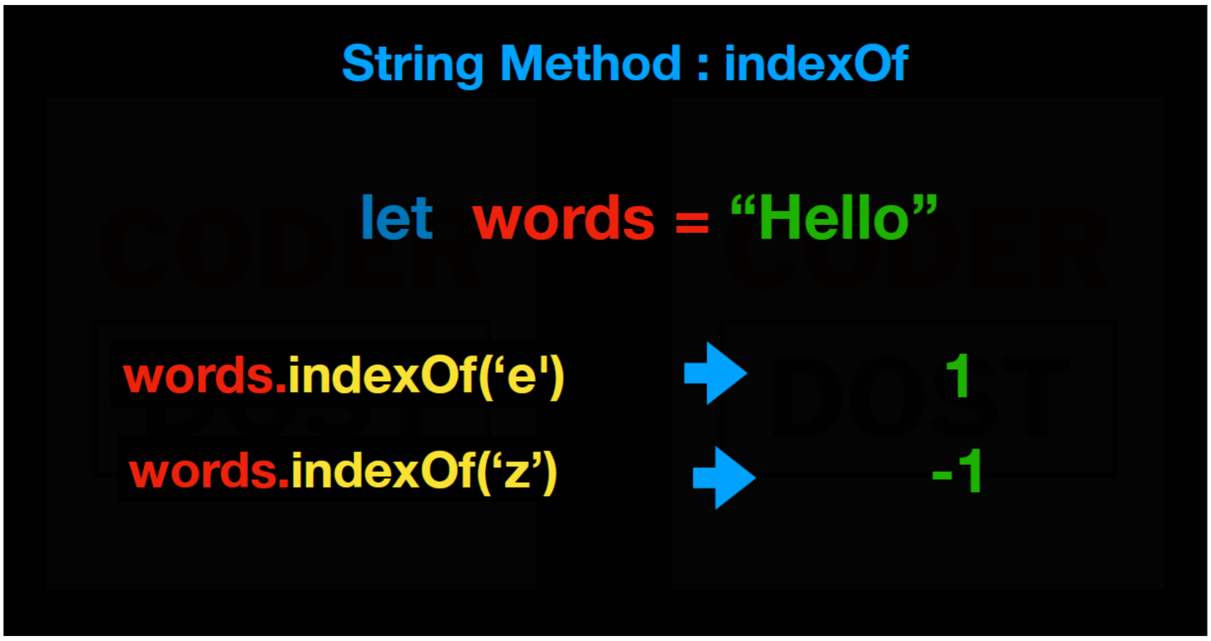
* What is a property ?? we will explain later



String Method* : upperCase / lowerCase

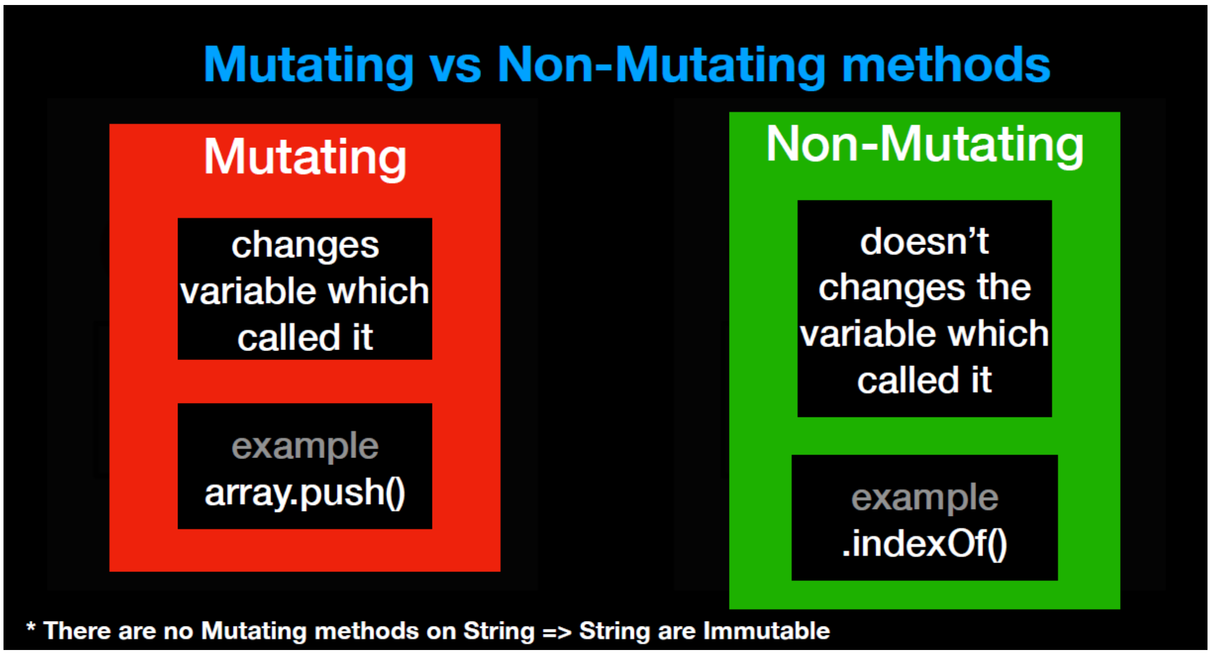
```
let  words = “Hello”

words.toUpperCase()  →  “HELLO”
words.toLowerCase()  →  “hello”
```

* What is a Method ?? we will explain later



String Method : indexOf

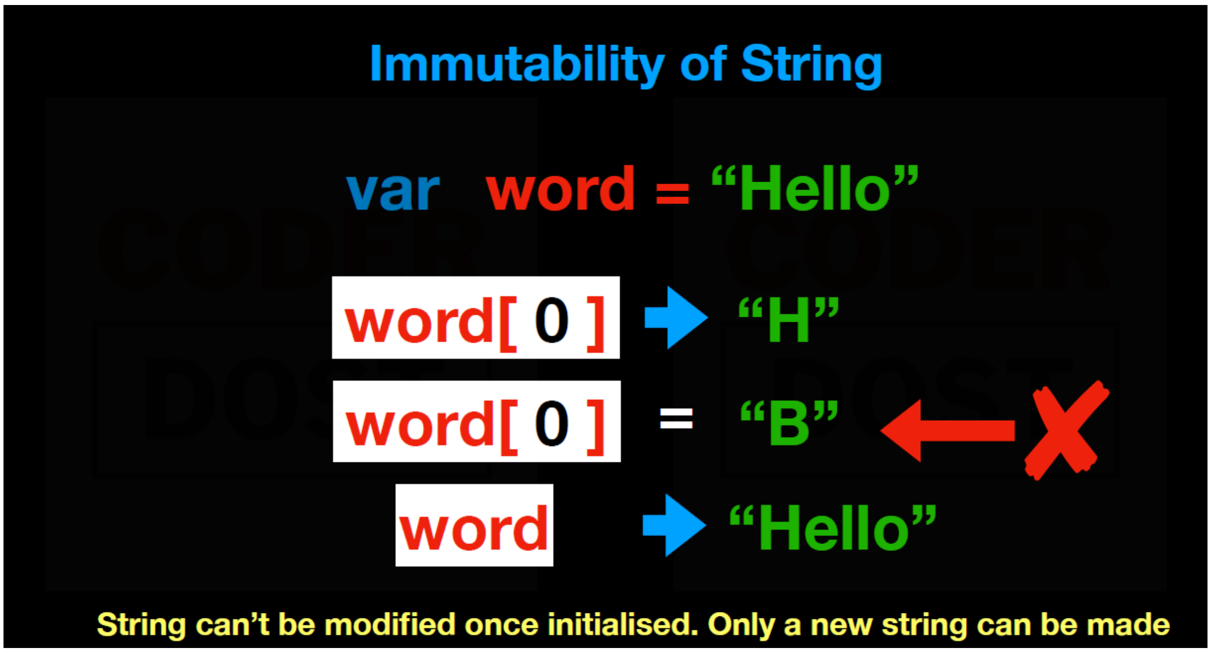
```
let  words = “Hello”

words.indexOf(‘e')    →    1
words.indexOf(‘z’)    →   -1
```



Mutating vs Non-Mutating methods

Mutating	Non-Mutating
changes variable which called it	doesn’t changes the variable which called it
example array.push()	example .indexOf()

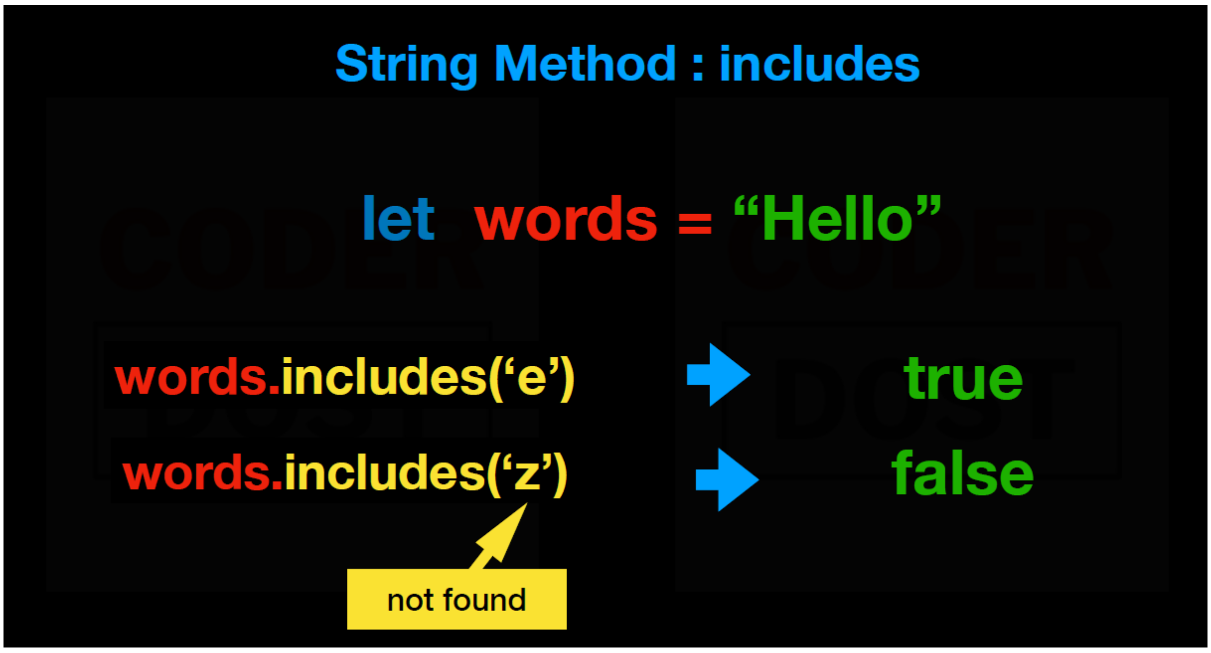
* There are no Mutating methods on String => String are Immutable



Immutability of String

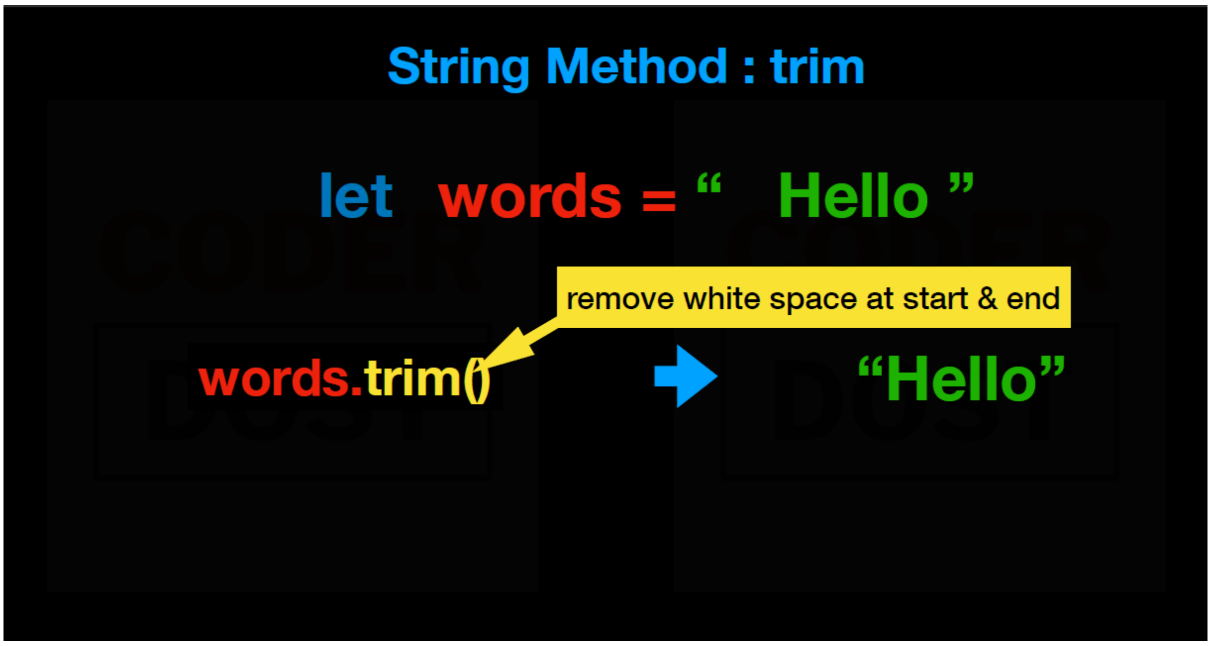
```
var  word = “Hello”

word[ 0 ]  →  “H”
word[ 0 ]  =  “B”  ←  ✗
word       →  “Hello”
```

String can’t be modified once initialised. Only a new string can be made



String Method : includes

```
let  words = “Hello”

words.includes(‘e’)   →   true
words.includes(‘z’)   →   false
```

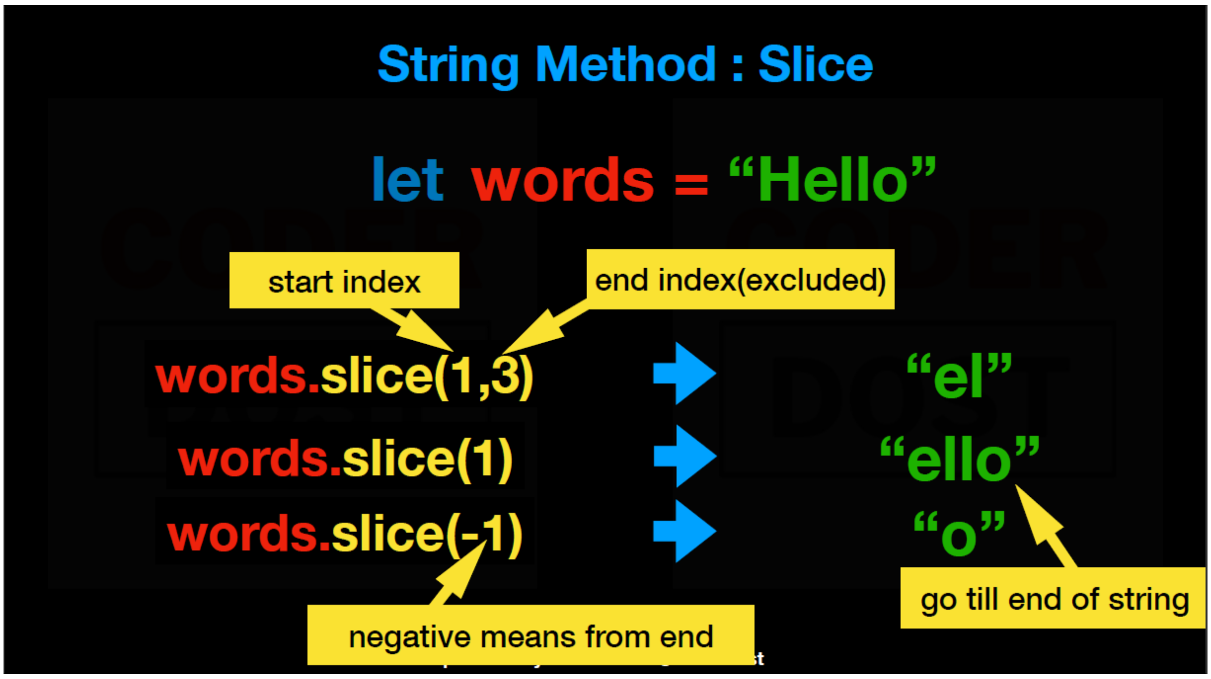


String Method : trim

```
let  words = “   Hello ”
```

remove white space at start & end

```
words.trim()   →   “Hello”
```



String Method : Slice

```
let  words = “Hello”
```

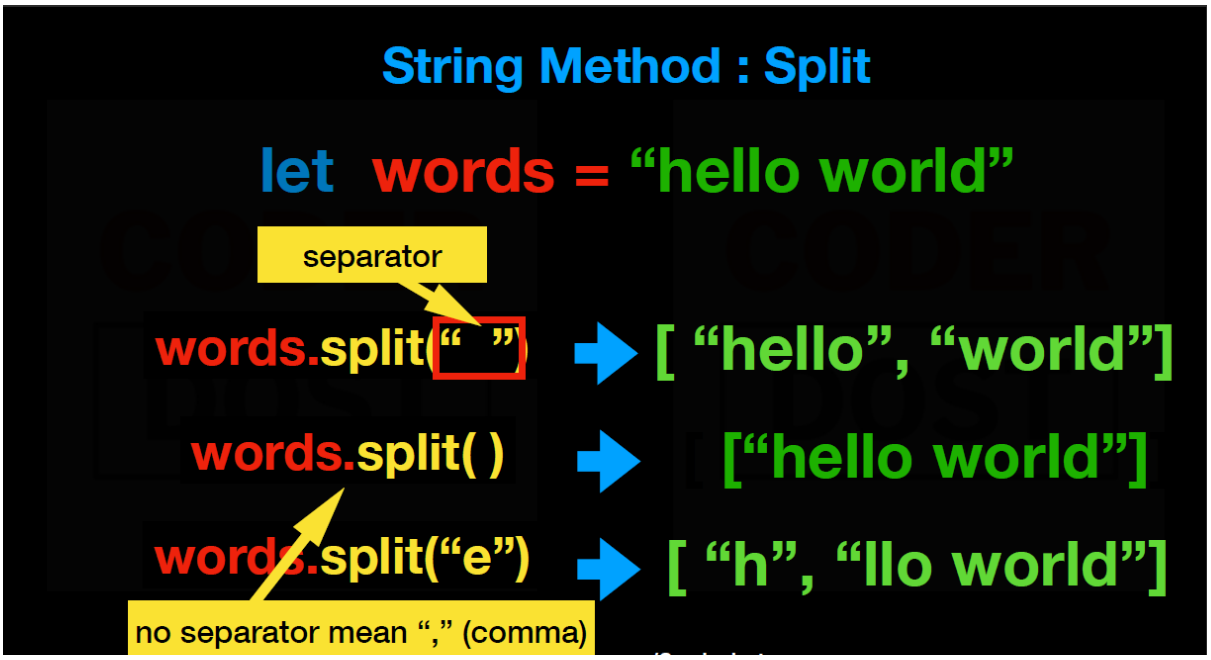
start index

end index(excluded)

```
words.slice(1,3)    →   “el”
words.slice(1)      →   “ello”
words.slice(-1)     →   “o”
```

negative means from end

go till end of string



String Method : Split

let words = “hello world”

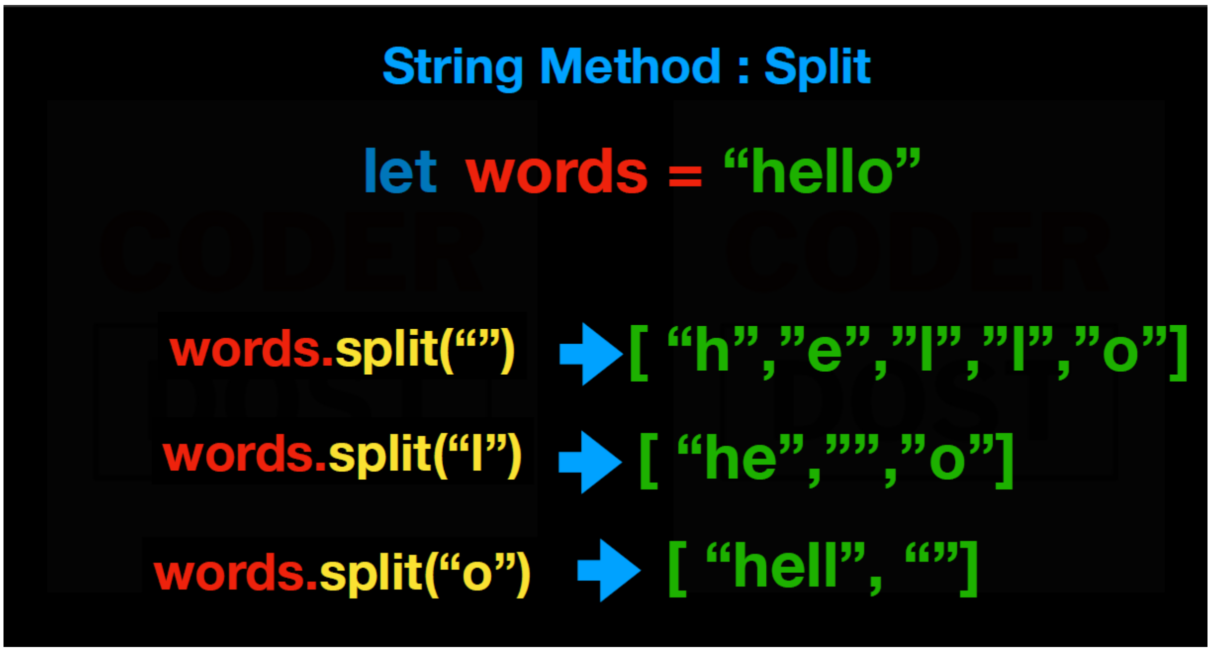
separator

words.split(“ ”) → [“hello”, “world”]

words.split( ) → [“hello world”]

words.split(“e”) → [“h”, “llo world”]

no separator mean “,” (comma)



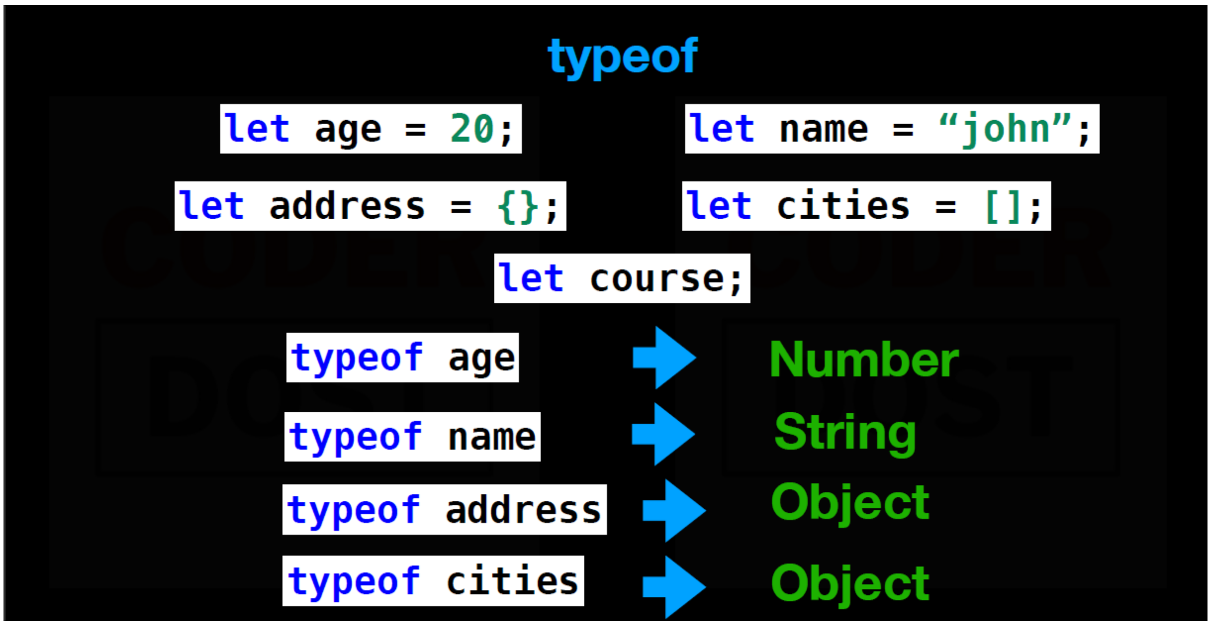
String Method : Split

let words = “hello”

words.split(“”) → [“h”,”e”,”l”,”l”,”o”]

words.split(“l”) → [“he”,””,”o”]

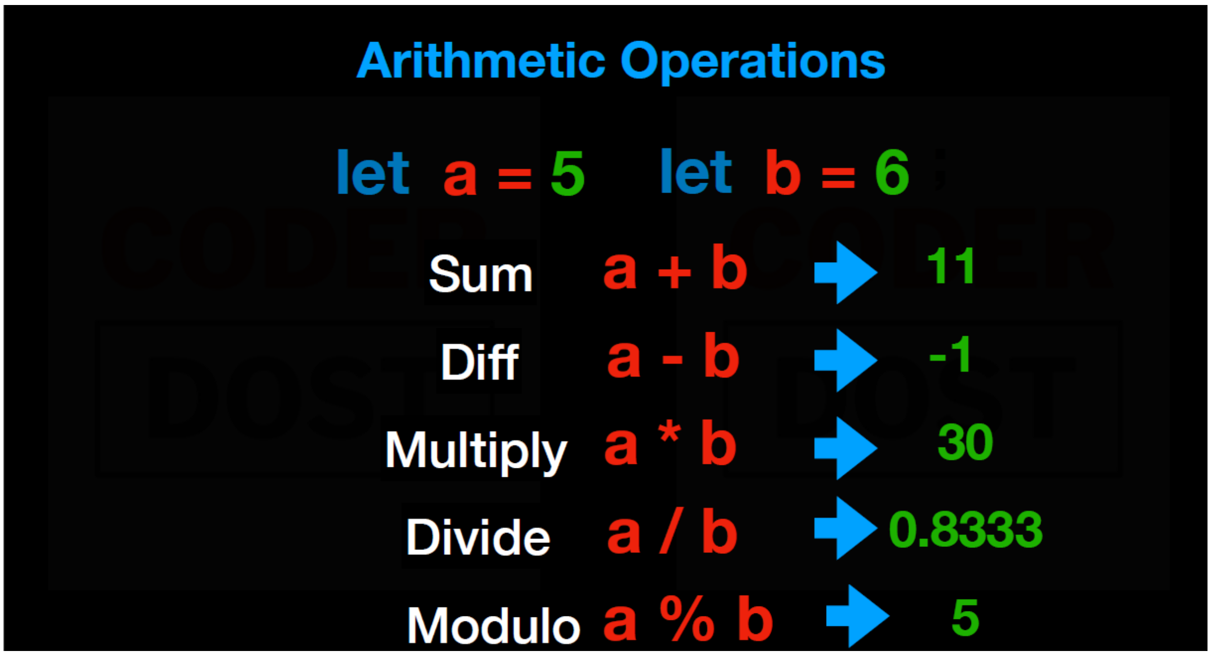
words.split(“o”) → [“hell”, “”]



typeof

```
let age = 20;          let name = “john”;
let address = {};      let cities = [];
let course;
```

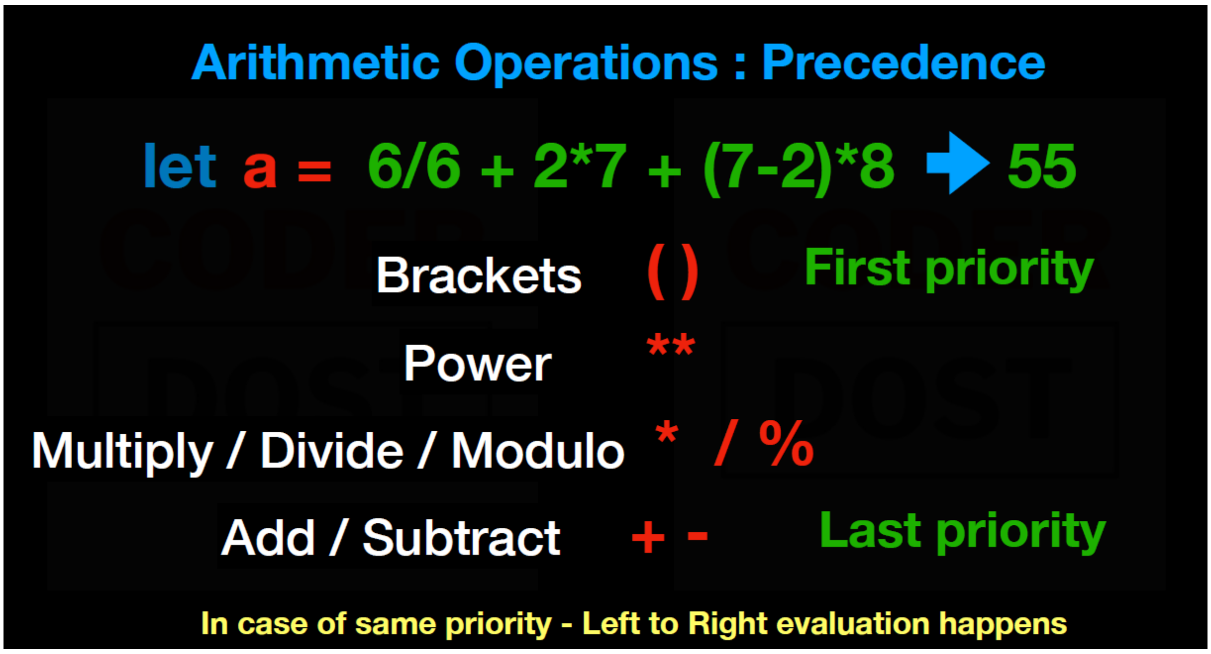
`typeof age`	Number
`typeof name`	String
`typeof address`	Object
`typeof cities`	Object



Arithmetic Operations

let a = 5 let b = 6 ;

Sum	a + b	11
Diff	a - b	-1
Multiply	a * b	30
Divide	a / b	0.8333
Modulo	a % b	5

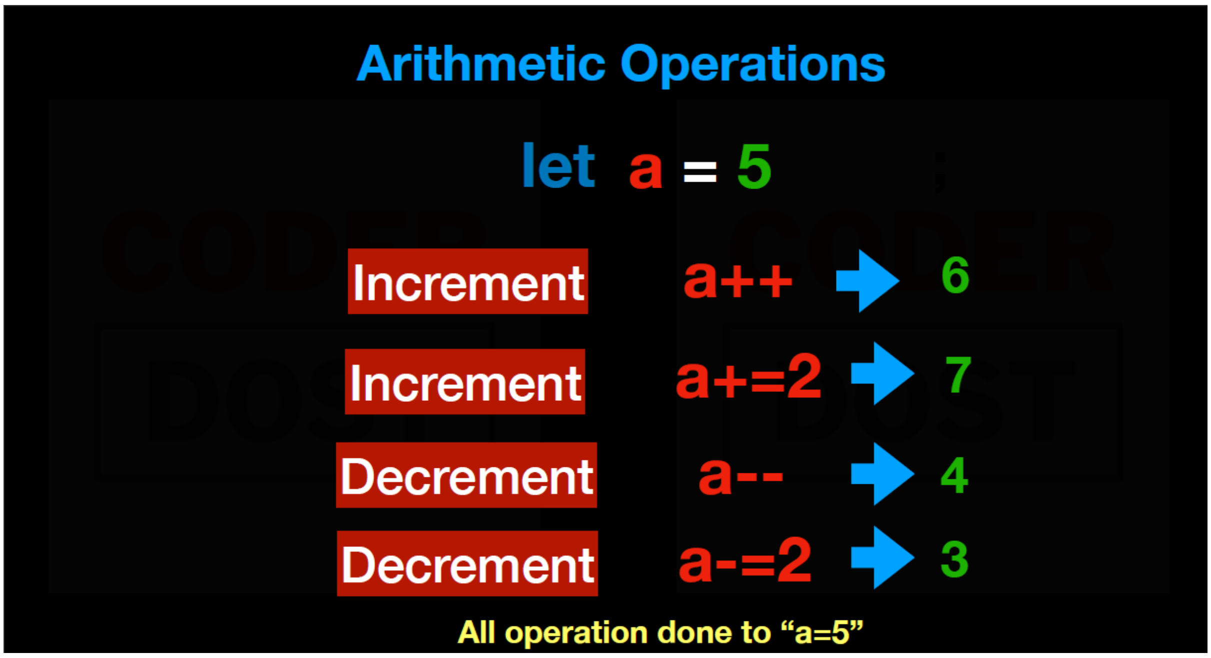


Arithmetic Operations : Precedence

```
let a = 6/6 + 2*7 + (7-2)*8 → 55
```

Brackets	( )	First priority
Power	**	
Multiply / Divide / Modulo	* / %	
Add / Subtract	+ -	Last priority

In case of same priority - Left to Right evaluation happens

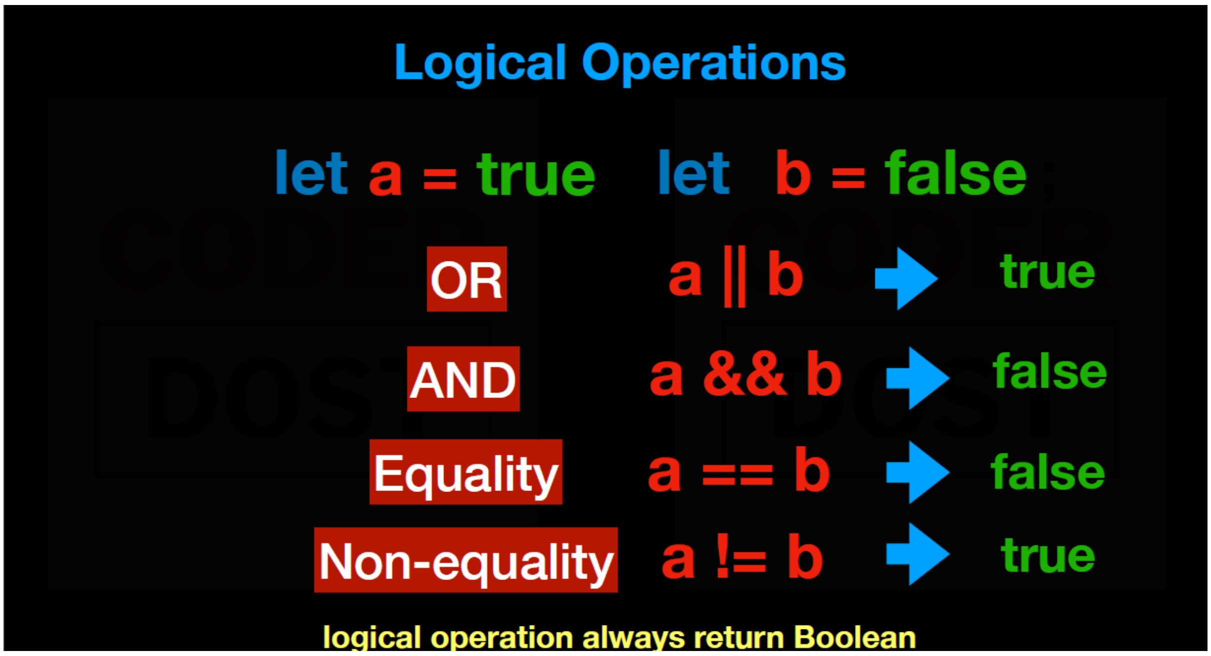


Arithmetic Operations

```
let a = 5
```

Increment	a++	→ 6
Increment	a+=2	→ 7
Decrement	a--	→ 4
Decrement	a-=2	→ 3

All operation done to “a=5”

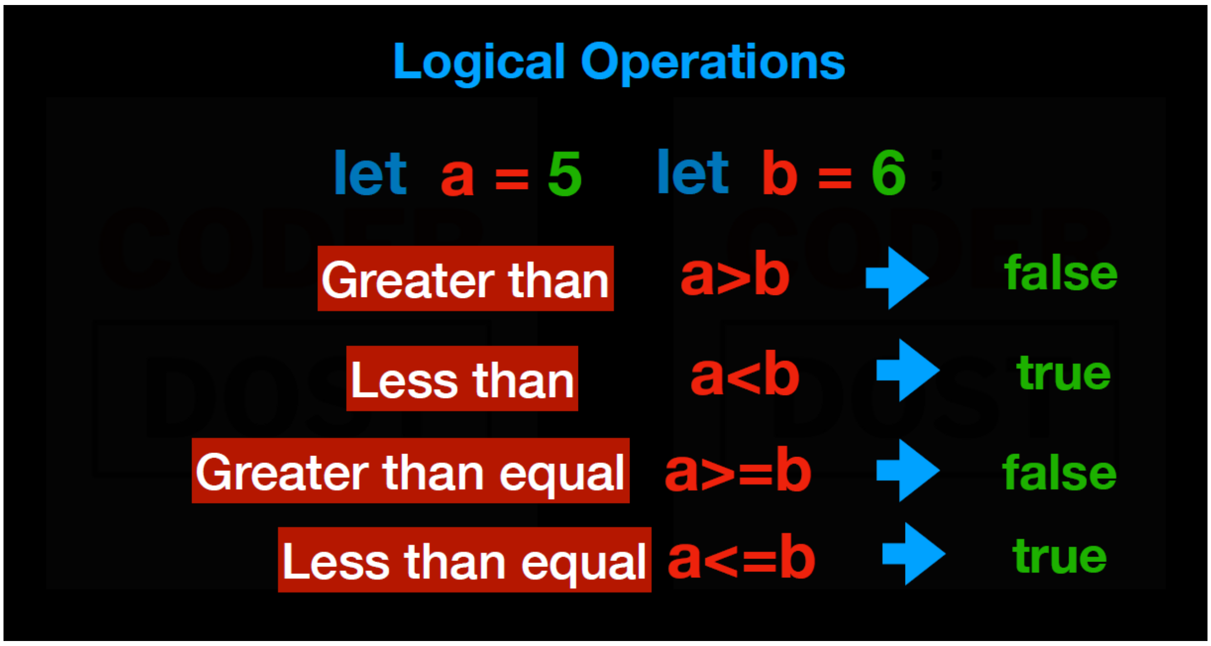


Logical Operations

let a = true let b = false

OR	a || b	➡	true
AND	a && b	➡	false
Equality	a == b	➡	false
Non-equality	a != b	➡	true

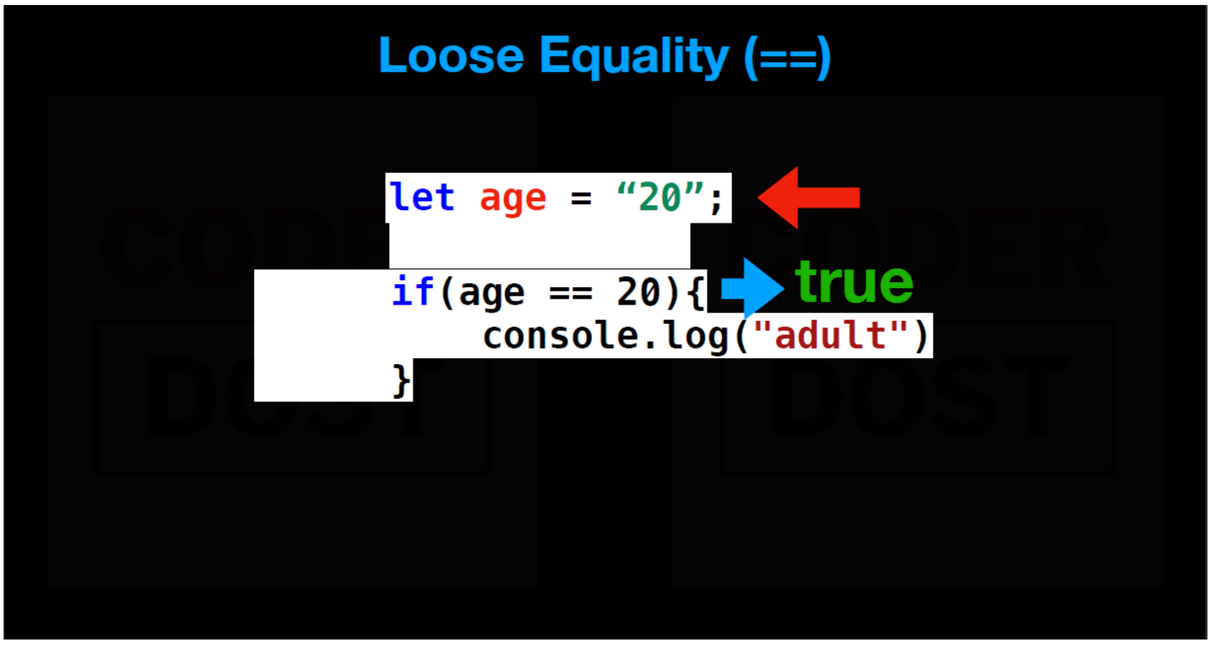
logical operation always return Boolean



Logical Operations

let a = 5 let b = 6

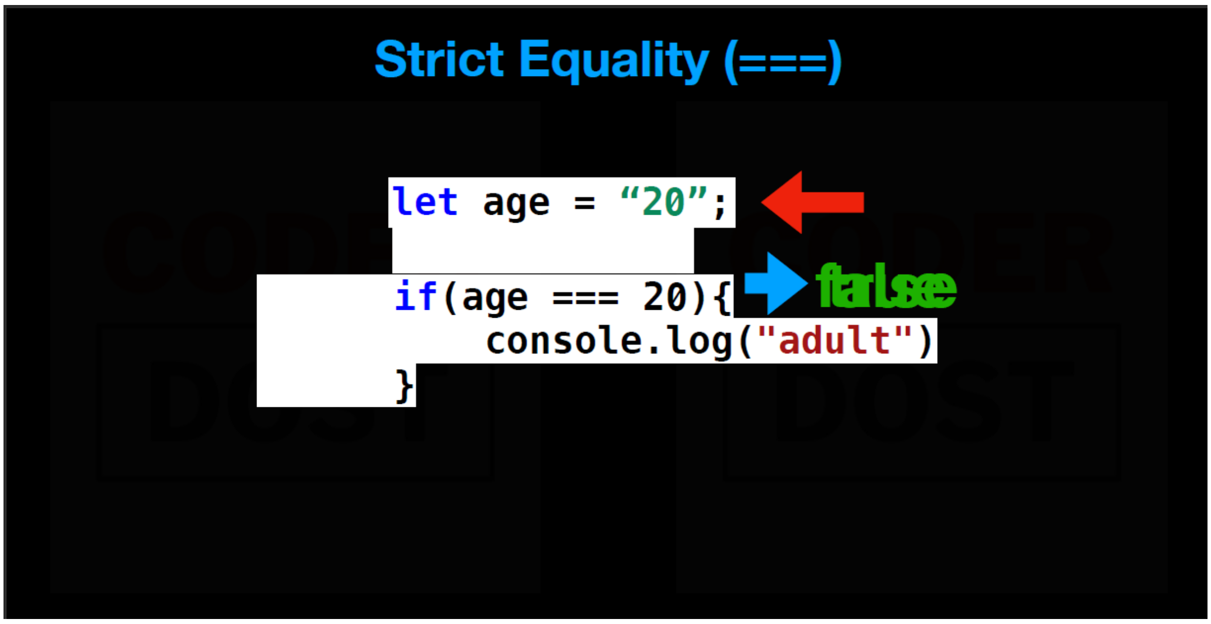
Greater than	a>b	➡	false
Less than	a<b	➡	true
Greater than equal	a>=b	➡	false
Less than equal	a<=b	➡	true



Loose Equality (==)

```
let age = “20”;

if(age == 20){
    console.log("adult")
}
```

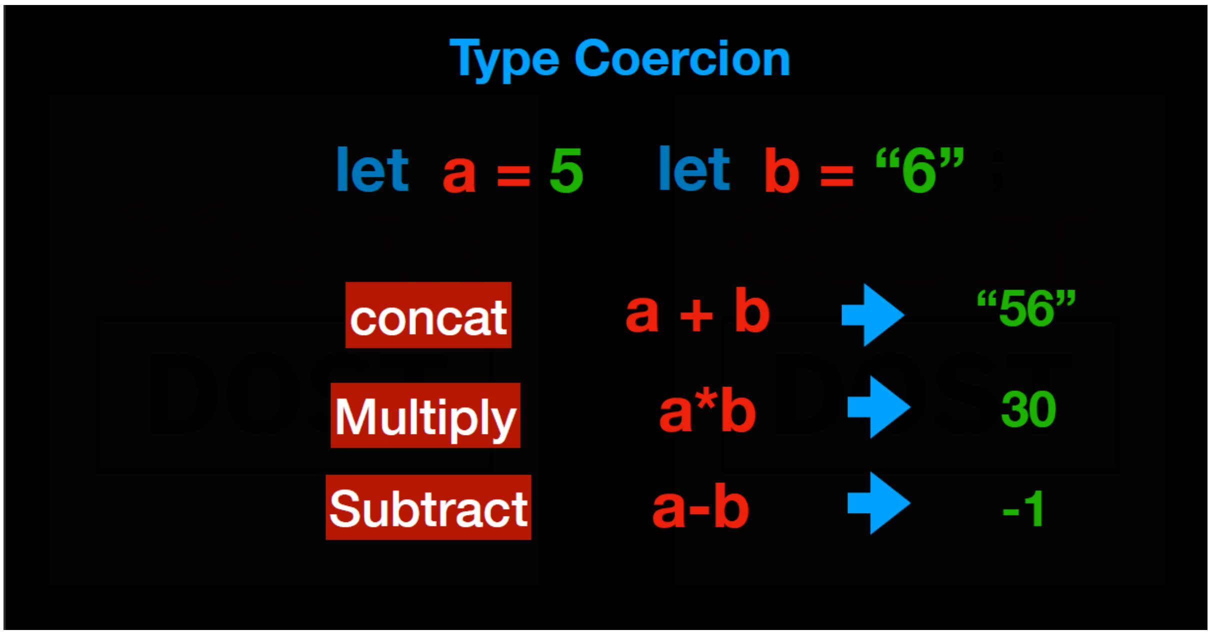
true



Strict Equality (===)

```
let age = “20”;

if(age === 20){
    console.log("adult")
}
```

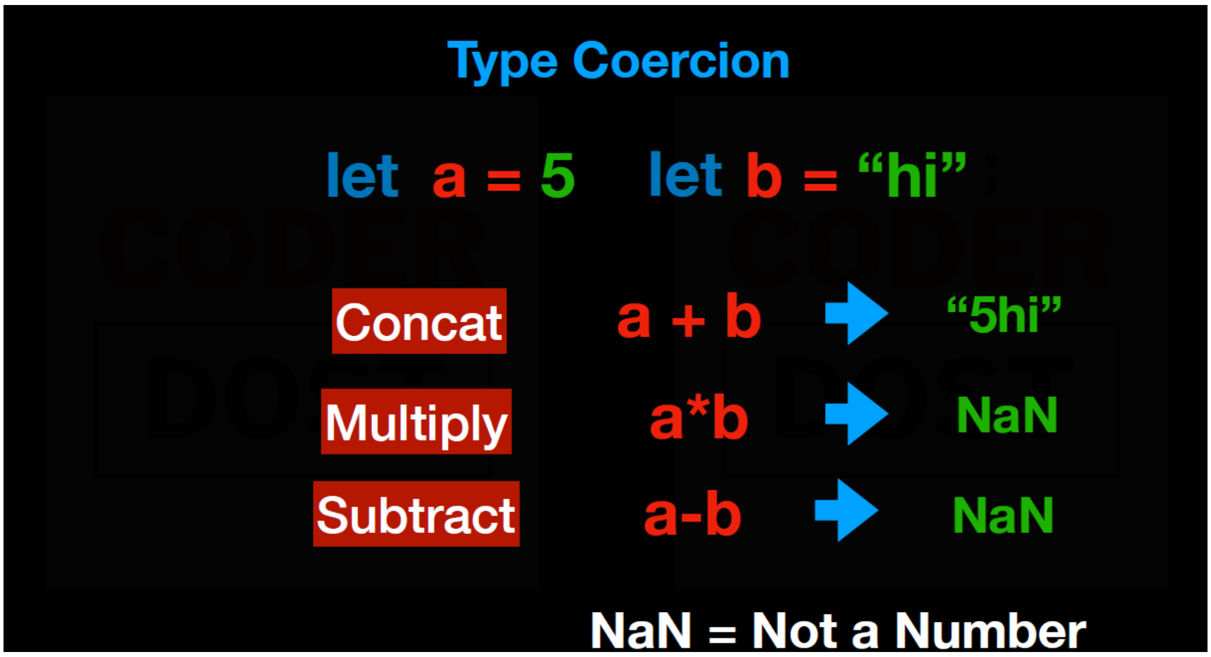
false



Type Coercion

let a = 5 let b = “6” ;

concat	a + b	➡	“56”
Multiply	a*b	➡	30
Subtract	a-b	➡	-1

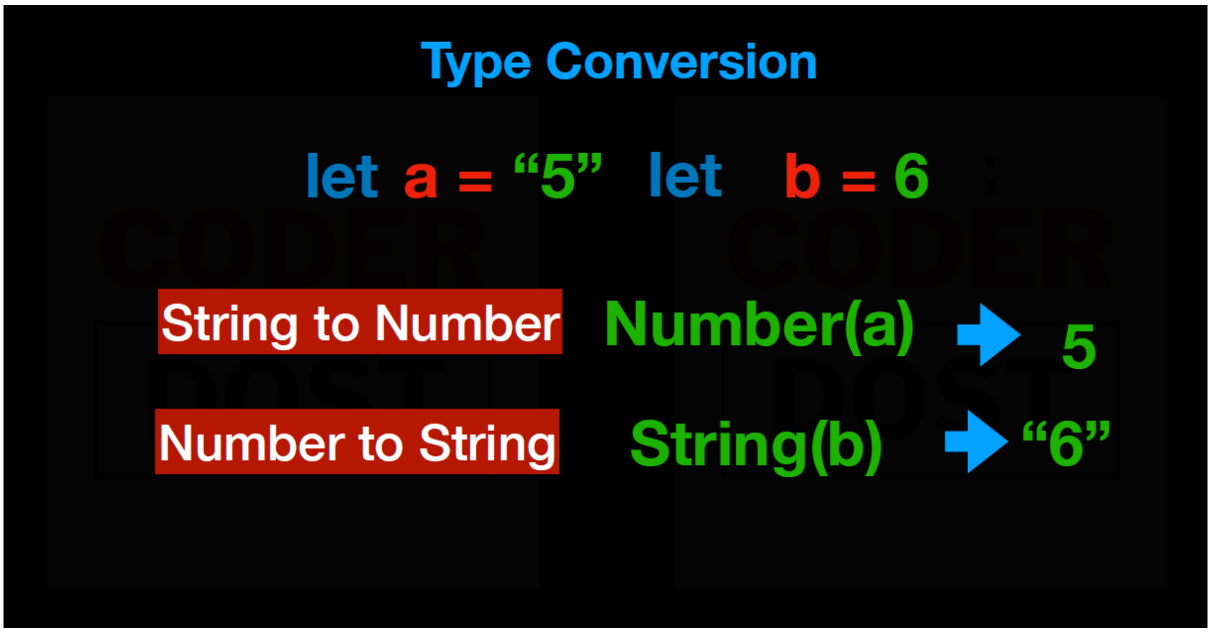


Type Coercion

let a = 5 let b = “hi” ;

Concat	a + b	➡	“5hi”
Multiply	a*b	➡	NaN
Subtract	a-b	➡	NaN

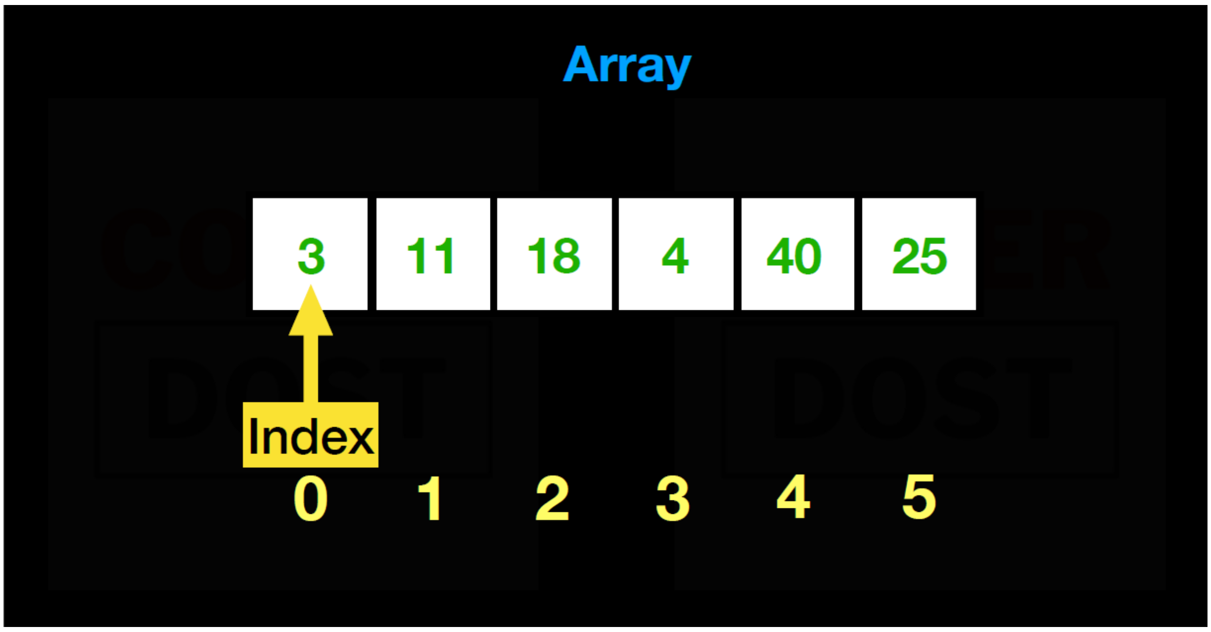
NaN = Not a Number



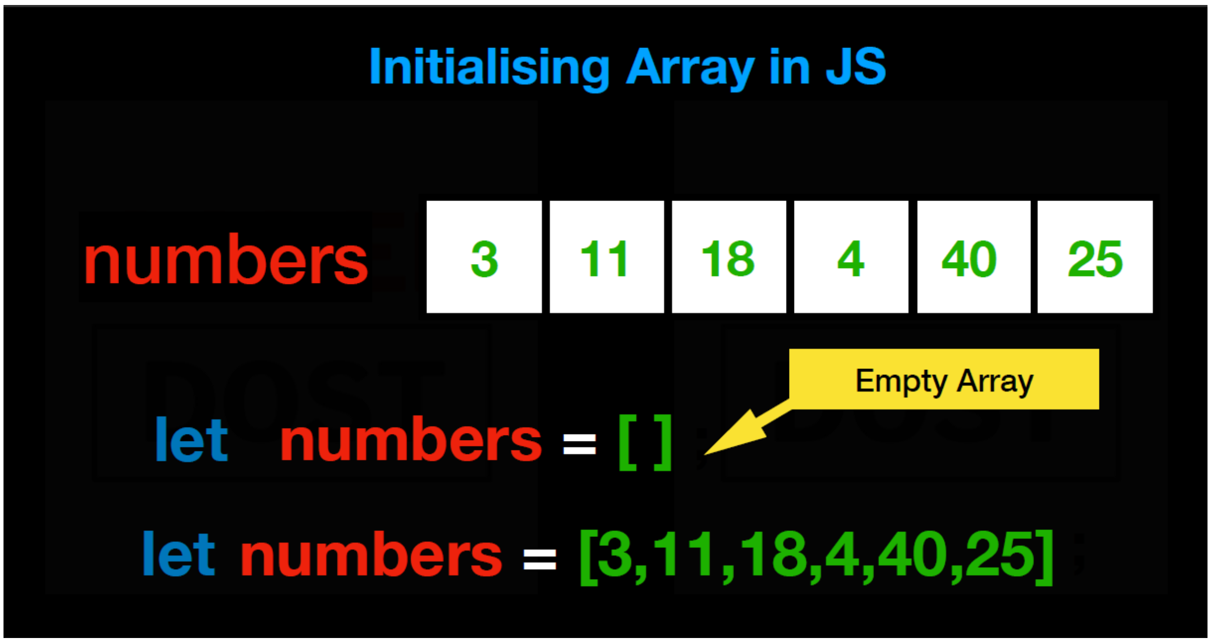
Type Conversion

let a = “5” let b = 6 ;

String to Number	Number(a)	→ 5
Number to String	String(b)	→ “6”



Array



Initialising Array in JS

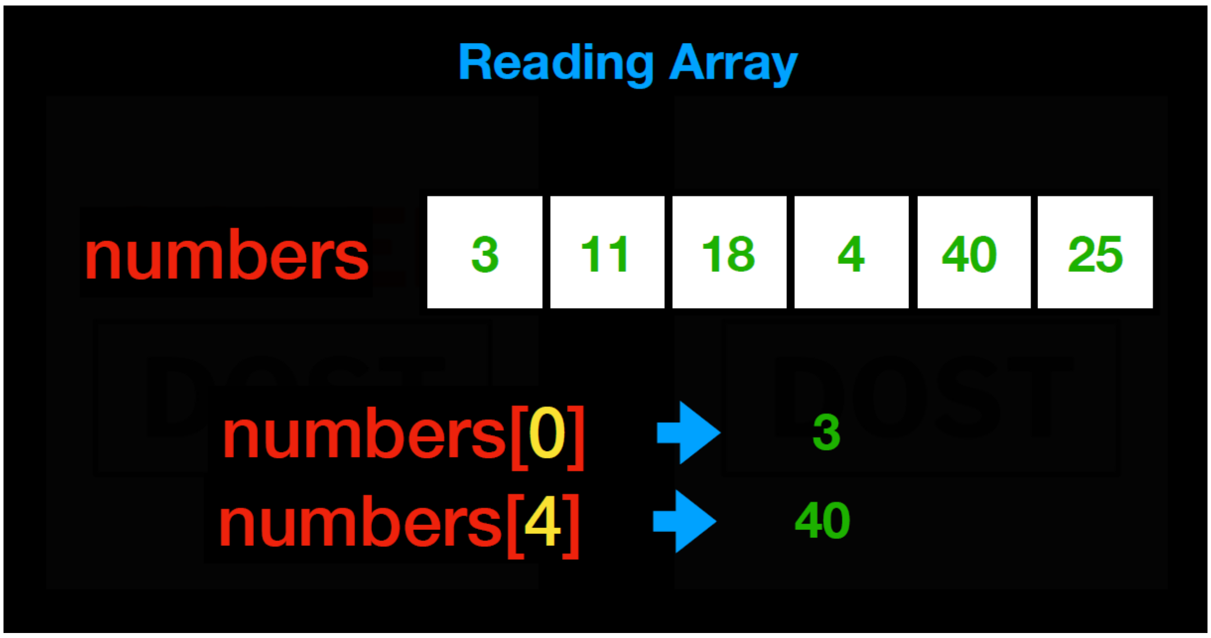
numbers

3	11	18	4	40	25

```
let  numbers = [ ]
```

Empty Array

```
let numbers = [3,11,18,4,40,25]
```



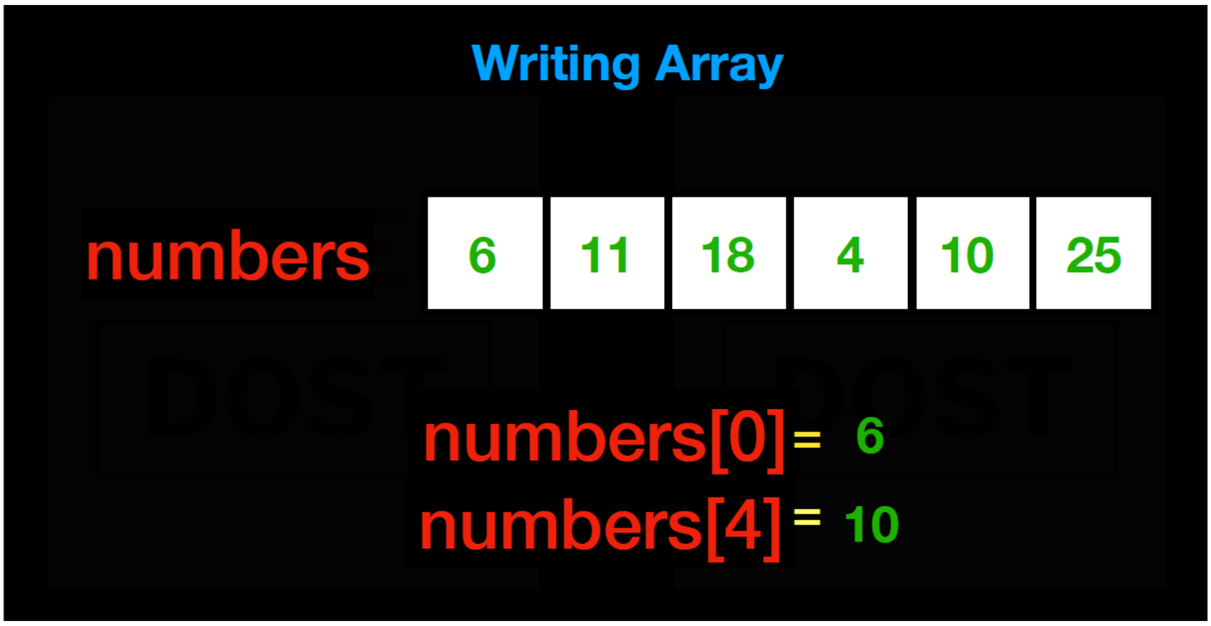
Reading Array

numbers

3	11	18	4	40	25

numbers[0] ➔ 3

numbers[4] ➔ 40

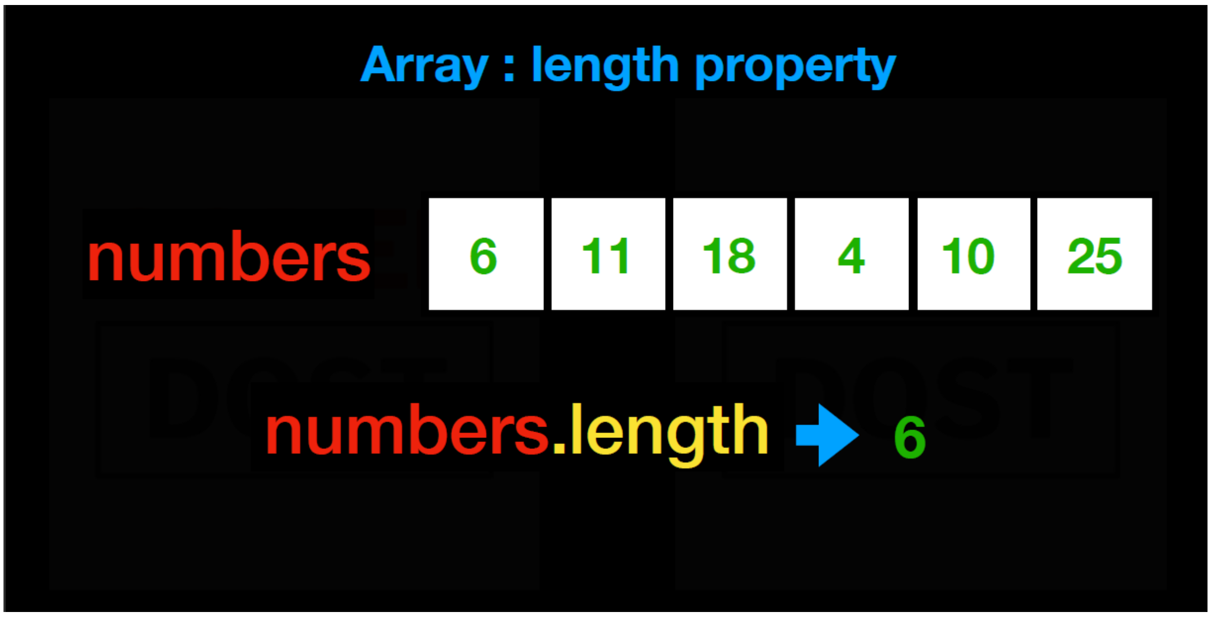


Writing Array

numbers	6	11	18	4	10	25

numbers[0]= 6

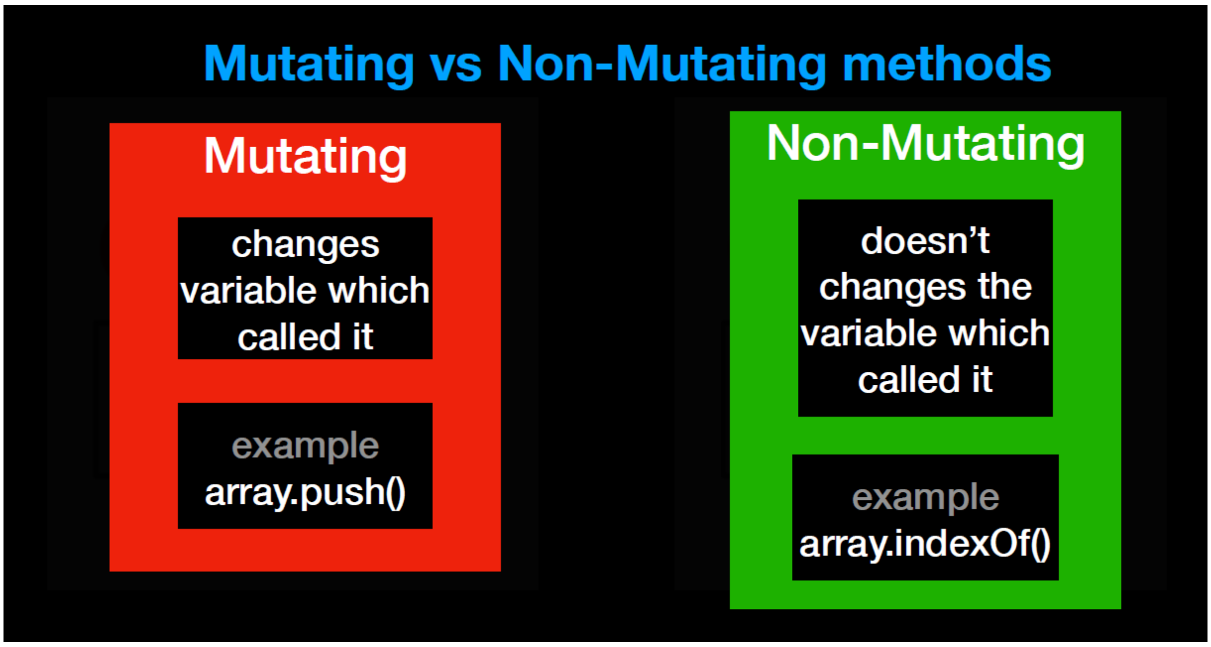
numbers[4]= 10



Array : length property

numbers	6	11	18	4	10	25

numbers.length $\rightarrow$ 6



Mutating vs Non-Mutating methods

Mutating

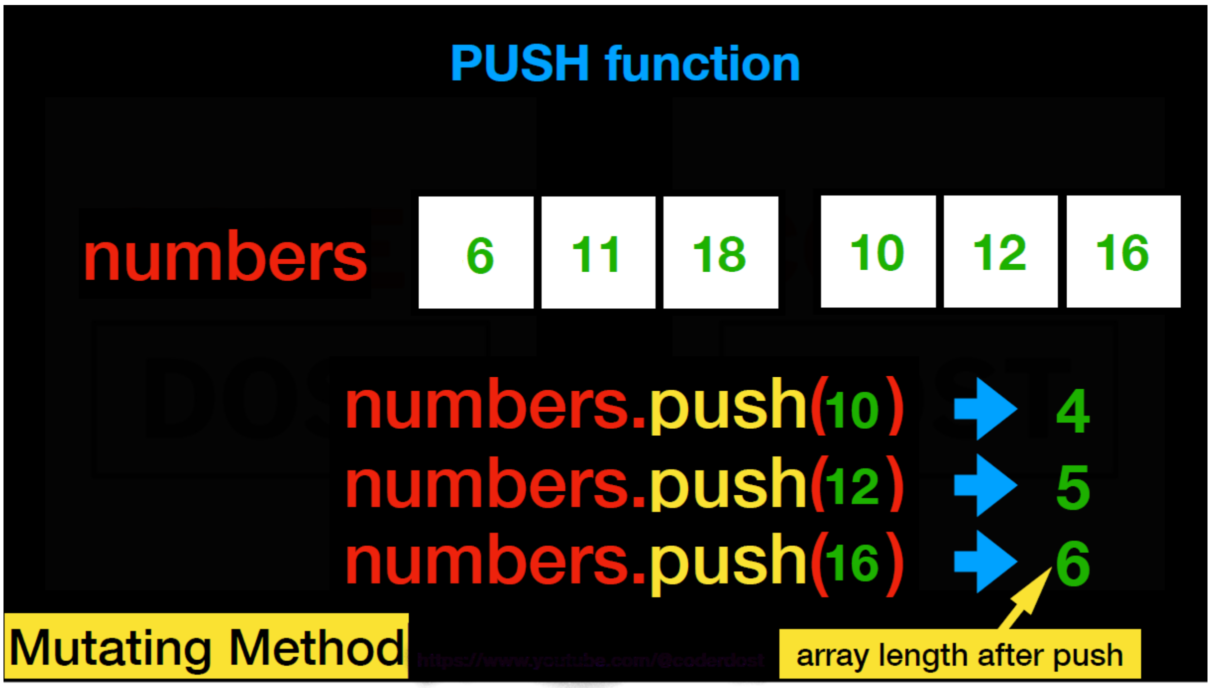
changes variable which called it

example
array.push()

Non-Mutating

doesn't changes the variable which called it

example
array.indexOf()



PUSH function

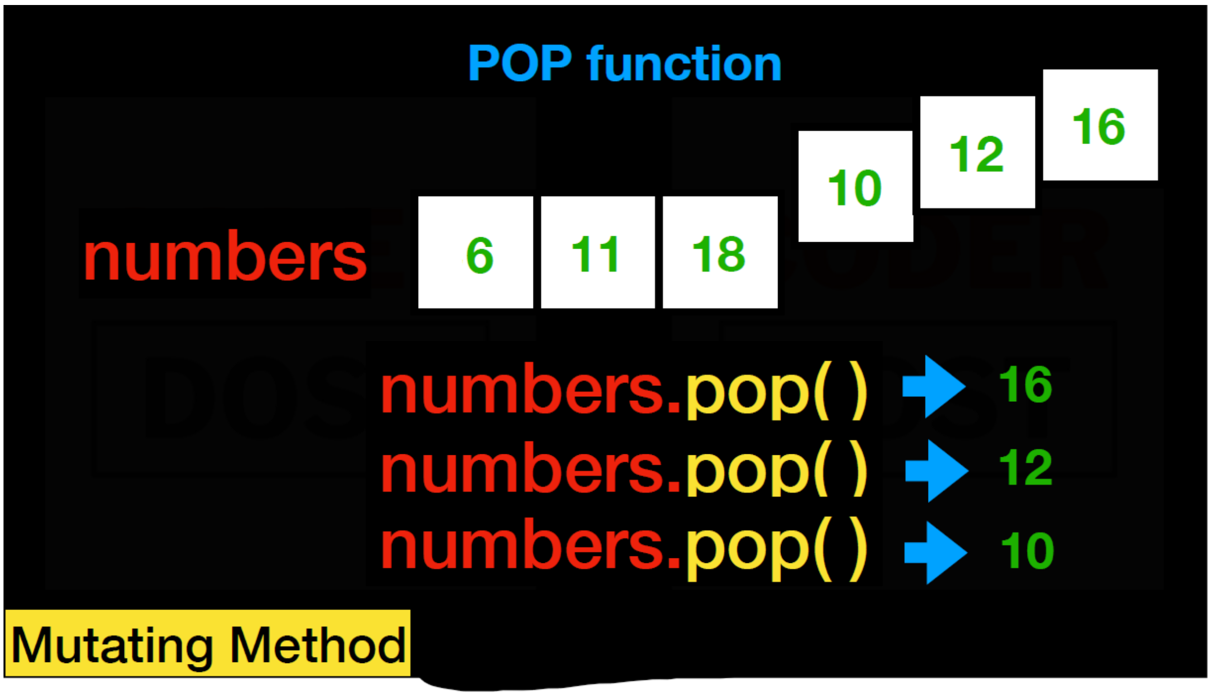
numbers

6	11	18	10	12	16

```
numbers.push(10) → 4
numbers.push(12) → 5
numbers.push(16) → 6
```

array length after push

Mutating Method

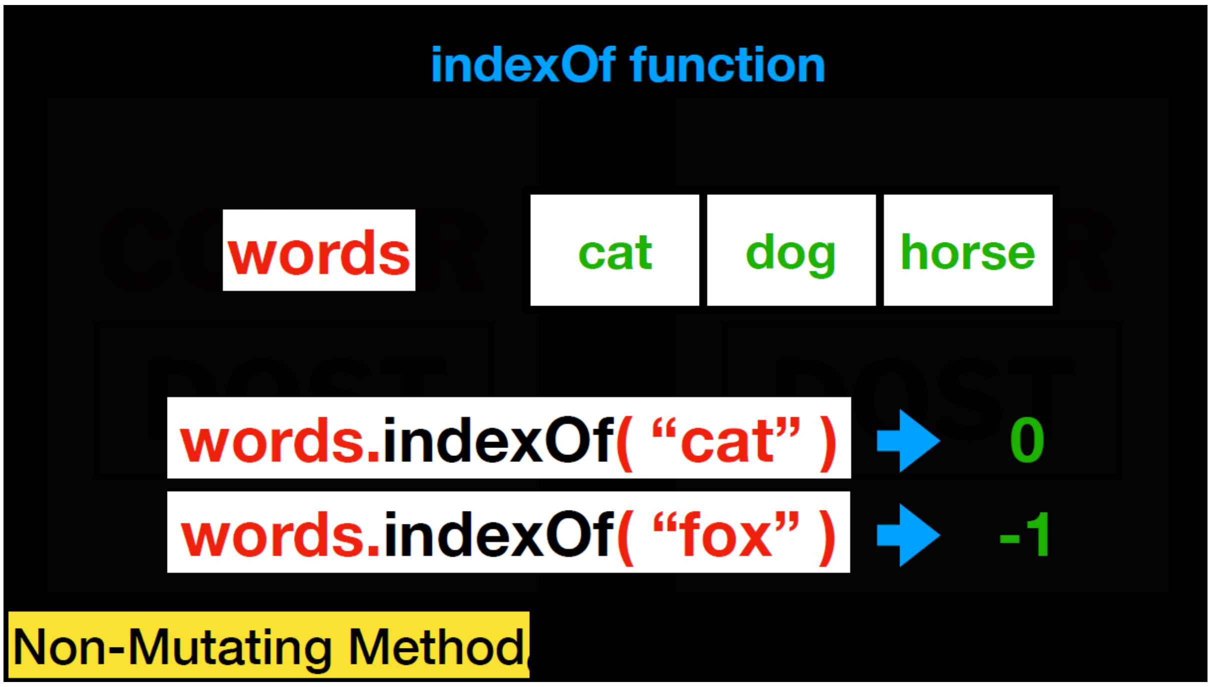


POP function

numbers 6 11 18 10 12 16

```
numbers.pop( ) → 16
numbers.pop( ) → 12
numbers.pop( ) → 10
```

Mutating Method

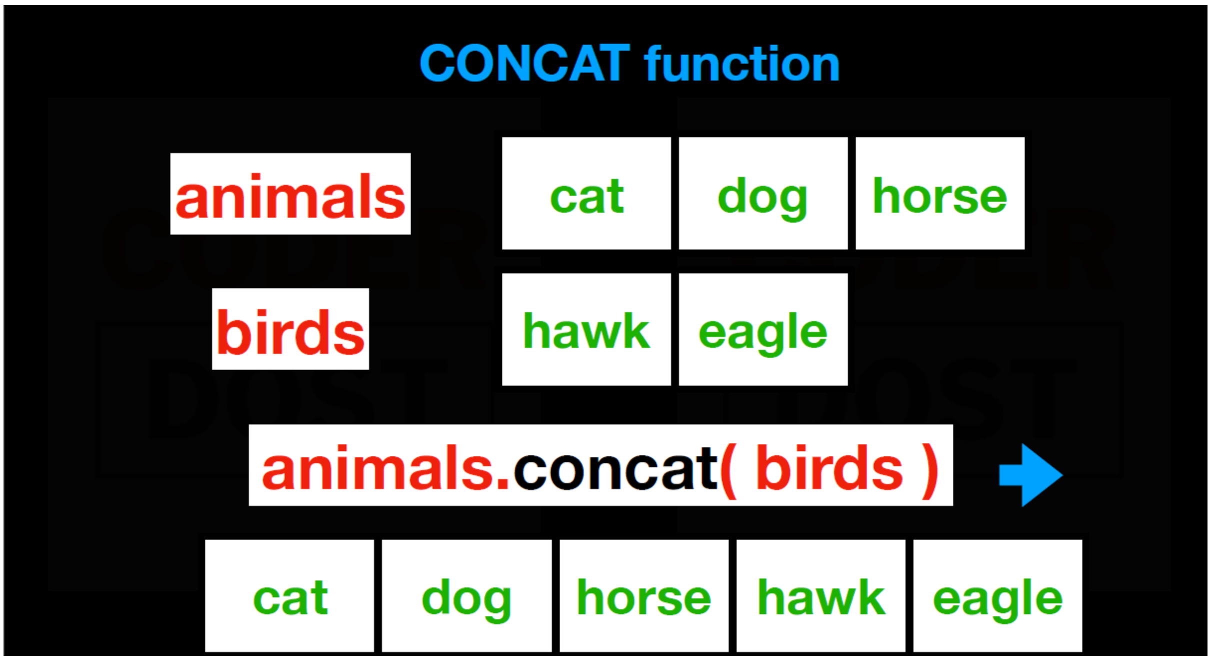


indexOf function

words cat dog horse

```
words.indexOf( “cat” ) → 0
words.indexOf( “fox” ) → -1
```

Non-Mutating Method



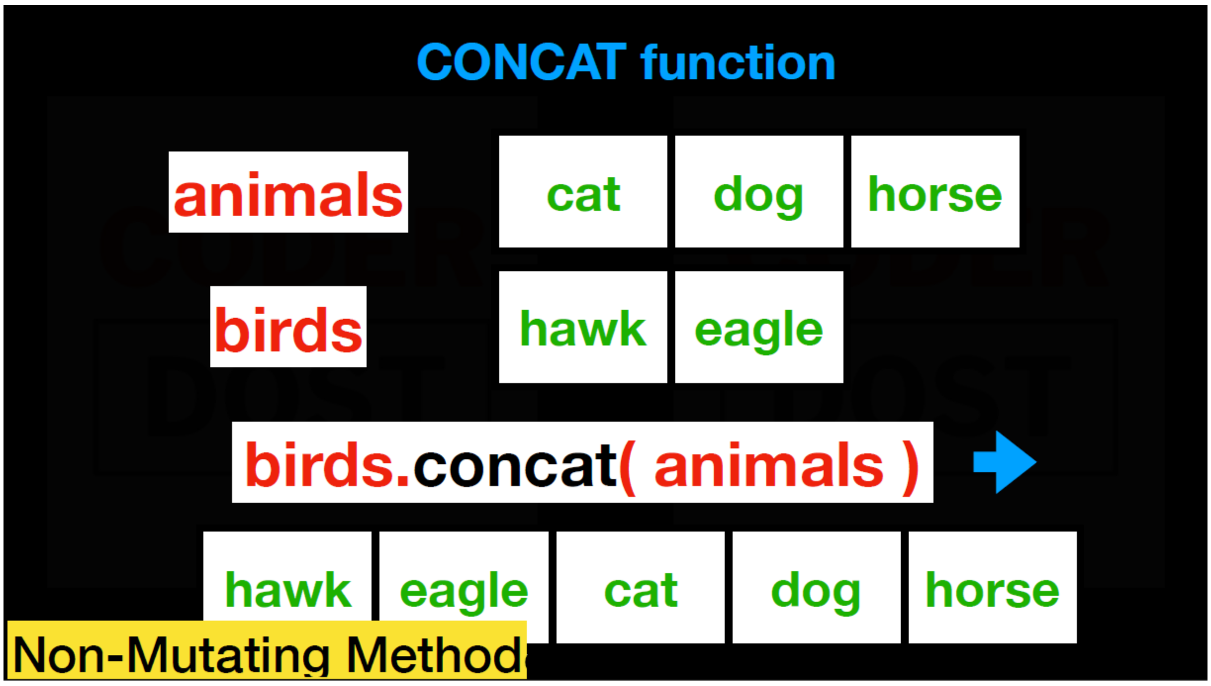
CONCAT function

animals: cat | dog | horse

birds: hawk | eagle

```
animals.concat( birds )
```

cat | dog | horse | hawk | eagle



CONCAT function

animals: cat | dog | horse

birds: hawk | eagle

```
birds.concat( animals )
```

hawk | eagle | cat | dog | horse

Non-Mutating Method