- “Javascript is synchronous single-threaded Language”
-

* Data Type (primitive(મૂળ) data type) :- *

String ⇒ “haresh”

Number ⇒ 2, 4, 5.6

Booleans ⇒ true & false

undefined

null

BigInt

symbol

(`) ⇒ backtick

(\) ⇒ backward slash

(/) ⇒ forward

(()) ⇒ round-bracket

([]) ⇒ square-bracket

({}) ⇒ curly-bracket

* Comment in JS :

*** single line comment :**

```
// only single line comment
```

*** multiple line comment :**

```
/*
   hello
   world
```

*/

(1) String :

- String Immutable હોય છે. string ને direct character by character change નથી કરી શકતો, તેની copy બનાવીને જ change કરી શકાય છે. variable ની જેમ string ની value પણ change નથી કરી શકાતી. existing માં એક character થી લઈને આખી string value change નથી કરી શકાતી, હંમેશા તેને re-assign કરાવીને જ value change કરવી પડે છે.
- string & array માં first character, element is on 0 and last element length -1.
- String literals are enclosed in

  single-quote or **double-quote** or **backtick**

  'Hello' "Hello" `${a}` or `hello`

* String Literal Template :-

Code :

```
let ['first name'] = 20;
console.log(['first name']);
```

Output :

20

How to Check Data Type :-

Code :

```
let age = 22;
let firstName = "haresh";
console.log(typeof age);
console.log(typeof firstName);
```

Output :

number
string

Convert Number to String :-

- variable સાથે ખાલી એક empty string round bracket ની અંદર add કરવાથી તે number string માં convert થઇ જાય છે.

Code :

```
let age = 22;
console.log(typeof (age + " "));
```

Output :

string (22)

Code :	Output :

let age = 22;
let Item = toString(age);
console.log(typeof item, age);
	22 (string)

Convert String to Number :-

- variable ની value આગળ simple +(plus) add કરી દેશો તો તે string માંથી number માં value change થઇ જશે.

Code :	Output :
let firstName = +“20”; console.log(typeof (firstName + “ ”));	number (2)
Code :	Output :

let age = “22”;
age = Number(age);
console.log(typeof age);
	22 (number)
	string (haresh)

Convert String to Array :-

Code :	Output :
let a = “haresh”; let b = a.split(“”); console.log(b);	[“h”, “a”, “r”, “e”, “s”, “h”] // convert to Array

print reverse string without loop :-

Code :	Output :
let a = “haresh-thumar”; let b = a.split(“”).reverse().join(“”); console.log(b);	ramuht-hserah //reverse print string

String Concatenation :-

Code :	Output :
let string1 = “haresh”; let string2 = “kumar”; let fullName = string1 + “ ” + string2; console.log(fullName);	haresh kumar

Convert String to Number & Two String Concatenation & addition (+) :-

Code :	Output :

let string1 = “17”;
let string2 = “10”;
let fullName = +string1 + +string2;
console.log(fullName);
	27

* type & typeof operator :

*** JavaScript માં દરેક value નીચેની type માં છે.**

- string ⇒ “haresh”
- number ⇒ 2, 4, 5, 7.5
- boolean ⇒ true & false
- The special values ⇒ null & undefined
- BigInt
- symbol

- non-object types ને સમસ્ત રીતે primitive types (મૂળ પ્રકારો) કહેવામાં આવે છે.
- જે value આપેલી છે, console માં તેની operator ની type (typeof) સાથે જો તમે use કરશો તો તેની type 'number', 'Boolean', 'undefined', 'object', 'string', 'symbol', કે નાની number string હશે તો તે output તમને return કરી દેશે.
- null એક object ની type છે.
- typeof null is the string 'object'.

* variable declaration :

• **var :**

- string, number, array, object કે function માં variable declare કરવા var keyword નો use ક્યારેય ના કરવો. જો તેના scope ની ખબર હોય તો જ use કરવું નહીંતર JavaScript વાળા use કરવાની ના પાડે છે higher order function માં var use ના કરેલ હોય તો તેના parent function માં search કરવા જાય છે. તેનાથી ના મળે તો parent ના parent function માં find કરવા જાય છે, variable પણ function keyword જ્યાં સુધી નહિ આવે ત્યાં સુધી તે search કરતું જ રહેશે. child function માં var નો scope જો function ની બહાર same name નો variable var keyword સાથે global declair કરેલો હશે તો તે print કરશે. પણ જો outer function ની અંદર બીજું inner function હશે તો તે inner function ના var નો scope તેના parent function સુધી જ રહે છે તો તે main function ની બહાર global declair હશે તો પણ તે print નહિ કરે. main function માં જ્યાંથી function keyword આવશે ત્યાં રોકાય જશે બહાર call નહિ કરે અને બીજા function માં same variable var સાથે declare કરેલો હશે તો તે પહેલા function નો variable બીજા function માં access નથી કરી શકતો. var માં એક variable ની value change કરવા માટે નીચે તમે બીજી વાર var keyword લખશો તે ચાલશે અને value update પણ કરી દેશે.

• **let :**

- let નો use primitives' value (string, number) માં જ કરવો. બીજે ક્યાંય use ના કરવો. કારણ કે let keyword થી બનાવેલ variable નો scope બે {} (curly bracket) ની વચ્ચે જ છે. એ પછી function માં declare કરવો કે block of code માં, બહાર ક્યાંય access ની અંદર કરી શકતો. function ની અંદર પણ જો તમારે value change કરવી હોય તો જ let keyword નો use કરવો નહીંતર const નો use કરવો. let keyword થી બે સરખા variable declare નથી કરી શકાતા, let માં એક variable ની નીચે value update કરાવવી હોય તો (without declare let keyword) only direct variable name + variable update value લખશો તો તે value change થઇ જશે. પણ બીજી line માં પણ let keyword use કરશો તો error આવી જશે. let keyword same name ના variable માં use નથી કરી શકાતો.

• **const :**

- આનો ઉપયોગ ખાલી array ને object માં જ કરવો. ક્યારેય var કે let નો use ના કરવો. તેનો scope declare કરાવેલા variable, array, object, કે function ની બહાર જતો નથી. const ને બીજી જગ્યાએ declare પણ નથી કરી શકાતો અને તેની value પણ નથી change કરી શકાતી. const ની value scope static જ રહે છે.

* identifier :

- identifier માં Unicode letter, digit, $(dollar) કે _(underscore) character નો સમાવેશ થાય છે. આ પહેલો character digit ના હોય શકે. કેટલીક વાર આપણે બીજી library use કરતા હોઈએ તો તે library માં ઘણી વાર $ કે _ નો already use કરેલો હોય તો error આવે છે અને match થવાની સંભાવના વધુ રહે છે, તેથી તેનો use ક્યારેય ના કરવો. અને identifier mean variable name, class name, object name, array name, કે function name માં always camelCase letter નો use કરવો.

* **identifier** માં ક્યારેય આવા **keyword use** ના કરવા :

break	case	catch	class	const	continue
debugger	default	delete	do	else	enum
export	extends	false	finally	for	function
if	import	in	instanceof	new	null
return	super	switch	this	throw	true
try	typeof	var	void	while	with

* **strict mode** માં આ બધા **keyword** નો **use** કરવાની મનાઈ છે :

implements	interface	let	package
protected	private	public	static

* **language** માં અત્યારે નીચેના નવા **keyword** પણ **add** થયા છે :

await	as	async	from	yield
get	of	set	target	

* Number :

- number value string માં store કરે છે, એને number માં store કરવા માટે number ની આગળ (+) લખવાથી તે value number માં store થાય છે.

- JavaScript માં minimum આટલા number જ store કરી શકીયે છીએ.
  MIN_SAFE_INTEGER (−253 + 1 or -9,007,199,254,740,991)

- અને maximum આટલા number જ store કરી શકીયે છીએ.
  MAX_SAFE_INTEGER (253 − 1 or 9,007,199,254,740,991)

- જો વધારે number store કરવા હોય તો એના માટે BigInt નો ઉપયોગ થઇ છે એ જાવાસ્ક્રિપ્ટ નું new feature છે.

```
let a = BigInt(616164165165164646556);
console.log(a);
```

*** Two Way Make the BigInt :**

- જો BigInt માં value આપવાથી તે BigInt ગણાય છે અને number ની પાછળ ખાલી small n add કરવાથી તે BigInt બની જાય છે.

(1) BigInt :	**(2) Number End to Add n :**
`let a = BigInt(123);` `console.log(a);`	`let a = 123n;` `console.log(a);`

- અપૂર્ણાંક(floating value) ભાગને કાઢી નાખવા માટે Math.trunc(x) અથવા Math.round(x) methode નો use કરવો.
- જો તમે શૂન્ય વડે ભાગો છો, તો પરિણામ Infinity or -Infinity છે. જો કે, 0/0 એ NaN ("not a number") છે.

```
counter += 10 // The same as counter = counter + 10
counter -= 10 // The same as counter = counter - 10

counter++ // The same as counter = counter + 1
```

counter - - // The same as counter = counter - 1
let riddle = counter++ //(post-increment) loop ચાલે પછી value increment થાય છે.
let enigma = ++counter //(pre-increment) loop ચાલે તે પહેલા જ value increment થઇ જાય છે.

Value	To Number	To String
The empty string ' '	0	' '
Any other string	NaN	Itself
false	0	'false'
true	1	'true'
null	0	'null'
undefined	NaN	'undefined'
The empty array [ ]	0	' '
Other arrays	NaN	The elements converted to strings and joined by commas, such as '1,2,3'
Objects	By default, NaN, but can be customized	By default, '[object Object]', but can be customized

- The values 0, NaN, null, undefined, and the empty string are each converted to **false**.
- all others to be **true**.

* null & undefined :

null is is false(0);
undefined is not declared value.

* for loop different () :

* traditional for loop using custom operation in array, object
* For of loop gives arrays & objects value
* for in loop gives arrays & objects index

* Array [ ] :

- Multiple Assign Value & string in one array. Array is mutable.
- const થી declare કરેલા array માં direct value change નથી કરાવી શકાતી, જો value change કરવી હોય તો method use કરીને કરી શકાય છે.
- array એક reference type છે.

* Check Array is Array ?

```
let fruits = ["apple", "mango", "graphes"];
console.log(typeof fruits);
```

* Check Deeply Array is Array ?

```
let fruits = ["apple", "mango", "graphes"];
console.log(Array.isArray(fruits));
```

* shift & unpush કરતાં pop & push method fast work કરે છે. કારણ કે shift & unshift માં પહેલા બધા element memory માં shift કરવા પડે છે અને પછી new element add થઇ છે.

* primitive value & references value store in memory :-

(1) primitive value :

- primitive type એ memory માં વધારે જગ્યા રોકતું નથી. primitive values માં બે variable અલગ અલગ location માં store થાય છે. એટલે આપણે value change કરીએ ત્યારે એક variable ની value update થાય છે, બીજા માં effect આવતી નથી. આ memory માં stack હોય છે. stack માં pointer હોય છે એ pointer માં value store થાય છે.

```
let num1 = 5;
```

```
let num2 = num1;
num1++;
console.log("num1 value is :", num1);    // 6
console.log("num2 value is :", num2);  // 5
```

(2) references value :

- references values બધા variable એક જ જગ્યાએ store થાય છે. આમાં બધા variable નું pointer એક જ address ઉપર રહે છે જેથી તમે value change કરશો તો બધાની variable ની value change થઈ જશે. આ બધા variable ની value memory માં heap માં store થાય છે.

```
let array1 = ["item1", "item2"];
array1.push("item3");
console.log("array1 is : ", array1);  // ["item1","item2","item3"]
console.log("array1 is : ", array2);  // ["item1","item2","item3"]
```

Example : how to clone array

```
//-----using slice method (faster optimize)------
let array1 = ["item1", "item2", "item3"];
let array2 = array1.slice(0);
console.log(array1);
console.log(array2);
//-----using concat method------
let array1 = ["item1", "item2", "item3"];
let array2 = [].concat(array1);
console.log(array1);
console.log(array2);
//-----using spread operator (using many people)------
let array1 = ["item1", "item2", "item3"];
let array2 = [...array1];
console.log(array1);
console.log(array2);
```

Example : example : add new item in array2

```
//-----example : add new item in array2------
   Example : 1
   let array1 = ["item1", "item2", "item3"];
   let array2 = array1.slice(0).concat(["item4", "item5"]);
   console.log(array1);
   console.log(array2);


    Example : 2
   let array1 = ["item1", "item2", "item3"];
   let array2 = [].concat(array1, "item4", "item5");
   console.log(array1);
   console.log(array2);
    Example : 3
   let array1 = ["item1", "item2", "item3"];
   let array2 = [...array1, "item4", "item5"];
   console.log(array1);
   console.log(array2);
    Example : 4
   let array1 = ["item1", "item2", "item3"];
   let oneMoreArray = ["item4", "item5"];
   let array2 = [...array1, ...oneMoreArray];
   console.log(array1);
   console.log(array2);
```

Example : array value print in uppercase & lowercase

```
//----- upperCase array ------
   Example : 1
   let item = ["apple", "banana", "grapes", "mango"];
   for(let a = 0; a < item.length; a++){
      console.log(item[a].toUpperCase());
   }
```

Example : 2

```
let item = ["apple", "banana", "grapes", "mango"];
let fruits = [];
for(let a = 0; a < item.length; a++){
    fruits.push(item[a].toUpperCase());
}
console.log(fruits);
```

//----- lowerCase array ------

Example : 1

```
let item = ["APPLE", "BANANA", "GRAPES", "MANGO"];
for(let a = 0; a < item.length; a++){
   console.log(item[a].toLowerCase());
}
```

Example : 2

```
let item = ["apple", "banana", "grapes", "mango"];
let fruits = [];
for(let a = 0; a < item.length; a++){
    fruits.push(item[a].toLowerCase());
}
console.log(fruits);
```

Example : while loop using print array

```
const item = ["apple", "banana", "grapes", "mango"];
const fruits = [];
let i = 0;
while(i < item.length){
    fruits.push(item[i].toUpperCase());
    I++;
}
console.log(fruits);
```

Example : for of loop using print array

```
const item = ["apple", "banana", "grapes", "mango"];
for(let fruit of item){
```

```
    console.log(fruit);
}

/*------- array value store in variable & print array using for of loop -------*/
const people = ["item1", "item2", "item3", "item4", ];
for (let index of people) {
    const var1 = index;
    console.log(var1);
}
```

Example : for loop using print array

```
const item = ["apple", "banana", "grapes", "mango"];
for(let a = 0; a < item.length; a++){
   console.log(item[a]);
}
```

Example : for in loop using print array

```
/*------ only print array index ------*/
const item = ["apple", "banana", "grapes", "mango"];
for(let fruit in item){
    console.log(fruit);
}

/*------ print array index with value ------*/
const item = ["apple", "banana", "grapes", "mango"];
for(let fruit in item){
    console.log(item[fruit]);
}
```

Example : array destructuring

```
/*----- print one array value in two variable & create  new array -----*/
```

Example : 1

```
const item = ["item1", "item2", "item3"];
let [a, b, c] = item;
console.log(a);
console.log(b);
console.log(c);
```

Example : 2

```
const item = ["item1", "item2", "item3"];
let a = item[0];
let b = item[1];
let c = item[2];
console.log(a);
console.log(b);
console.log(c);
```

Example : 3

```
const haresh = ["value1", "value2", "value3", "value4"];
let [myVar1, myVar2, ...item] = haresh;
console.log(myVar1);
console.log(myVar2);
console.log(item);
```

* Object {} :

- object પણ એક reference type જ છે.

Example : print object value in two method

```
/*----- access element in object (.) dot -----*/
const obj = {name: "haresh", age: 20};
obj.gender = "male";
console.log(obj);
```

```
/*----- access element in object ([" "]) bracket notation -----*/
```

Example : 1

```
const obj = {
    name: "haresh",
    age: 20,
    gender : "male"
};
console.log(obj["gender"]);
```

Example : 2

```
const obj = {
    name: "haresh",
    age: 20,
    "my hobbies" : ["singing", "dancing", "reading", "swimming"]
};
console.log(obj["my hobbies"]);
```

Example : 3

```
let key = "email";
const obj = {
    name: "haresh",
    age: 22,
    gender: "male",
}
obj[key] = "haresh@gmail.com"
console.log(obj);
```

Example : print object value in two method using for loop

```
/*---------- for in loop ----------*/
const obj = {
    name: "haresh",
    age: 20,
    "my hobbies" : ["singing", "dancing", "reading", "swimming"]
};
for(let haresh in obj){
    console.log(obj[haresh]);
}
```

```
/*---------- object.keys object literal ----------*/
```

Example : 1

```
const obj = {
    name: "haresh",
    age: 20,
    "my hobbies": ["singing", "dancing", "reading", "swimming"]
};
for (let key in obj) {
    console.log(`${key} : ${obj[key]}`);
}
```

Example : 2

```
const obj = {
    name: "haresh",
    age: 20,
    "my hobbies": ["singing", "dancing", "reading", "swimming"]
};
for (let key in obj) {
    console.log(key, ":", obj[key]);
}
```

Example : access key using Object.keys

Example : 1

```
const obj = {
    name: "haresh",
    age: 20,
    "my hobbies": ["singing", "dancing", "reading", "swimming"]
};
console.log(typeof (Object.keys(obj)));
const val = Array.isArray((Object.keys(obj)));
console.log(val);   //return true or false
```

Example : 2

//------ only print key name

```
const obj = {
```

```
    name: "haresh",
    age: 20,
    "my hobbies": ["singing", "dancing", "reading", "swimming"]
};
for(let key of Object.keys(obj)){
    console.log(key);
}
```

//------ only print key value

```
const obj = {
    name: "haresh",
    age: 20,
    "my hobbies": ["singing", "dancing", "reading", "swimming"]
};
for(let key of Object.keys(obj)){
    console.log(obj[key]);
}
```

Example : make a object using variable value

Example : 1

```
const key1 = "ovject1";
const key2 = "ovject2";
const value1 = "myValue1";
const value2 = "myValue2";
const obj = {
    [key1] : value1,
    [key2] : value2,
}
console.log(obj);
```

Example : 2

```
const key1 = "ovject1";
const key2 = "ovject2";
const value1 = "myValue1";
const value2 = "myValue2";
const obj = {};
```

```
obj[key1] = value1;
obj[key2] = value2;
console.log(obj);
```

* Spread operator :

- spread operator એ array કે object ને clone કરાવવા માટે અથવા આખો string, array, object ના element ને spread કરાવવા માટે use થાય છે.
- spread operator માં only Number iterable નથી. Number લખવા હોય તો ("") double quote ની અંદર લખવા પડે છે તો Number iterate થાય છે.

Example : Clone Array

```
/*---------- Clone Array using spread operator ----------*/
const array1 = [1, 2, 3];
const array2 = [4, 5, 6];
const newArray = [...array1, ...array2];
console.log(newArray);
or
const newObject = {...["item1", "item2"] }
console.log(newObject);
```

Example : Clone Object

```
/*---------- Clone Object using spread operator ----------*/
const obj1 = {
    fName: "haresh",
    hAge: 24,
    hBirthday: "21 Nov, 1997"
};
const obj2 = {
    lName: "parth",
    pAge: 22,
    pBirthday: "i Don't Know"
};
const newObj = {...obj1, ...obj2 };
console.log(newObj);
```

Example : Clone String

```
/*----------- Clone string using spread operator -----------*/
  const str1 = "haresh";
  const str2 = "thumar";
  const newStr = [...str1, ...str2];
  console.log(newStr);
  or
  const newObject = {... "haresh" }
  console.log(newObject);
```

Example : Clone Number

```
/*----------- Clone number using spread operator -----------*/
  const str1 = "01234";
  const str2 = "56789";
  const newStr = [...str1, ...str2];
  console.log(newStr);
  or
  const newObject = {... "22446688" }
  console.log(newObject);
```

Example : make a variable in using array item name

Object key is ByDefault Variable :

```
const haresh = {
    name: "parth",
    surName: "savaliya",
}
const { name, surName} = haresh;
console.log(name, surName);
```

Extra add Variable Name :

```
const haresh = {
    name: "parth",
    surName: "savaliya",
```

```
}
const { name: var1, surName: var2 } = haresh;
console.log(var1, var2);
```

Object key is ByDefault Variable with using spread operator :

```
const haresh = {
    name: "parth",
    surName: "savaliya",
    Year: 1968,
    place: "kashmir" ,
}
const { name, surName, ...extraKey} = haresh;
console.log(name, surName);
console.log(extraKey);
```

Example : store multiple object in one array & print multiple & particular item using for of loop

```
const user = [
    { userId: 1, firstName: "haresh", gender: "male" },
    { userId: 2, firstName: "rushabh", gender: "male" },
    { userId: 3, firstName: "komal", gender: "female" },
];
for (let item of user) {
    console.log(item);
    console.log(item.userId);
    console.log(item.firstName);
    console.log(item.gender);
};
```

Example : nested destructuring

```
//-------- store object value in variable after print all array object --------
const user = [
    { userId: 1, firstName: "haresh", gender: "male" },
    { userId: 2, firstName: "rushabh", gender: "male" },
    { userId: 3, firstName: "komal", gender: "female" },
];
const [user1, user2, user3] = user;
```

```
    console.log(user1, user2, user3);

*/-------- print particular object value --------*/
    const user = [
        { userId: 1, firstName: "haresh", gender: "male" },
        { userId: 2, firstName: "rushabh", gender: "male" },
        { userId: 3, firstName: "komal", gender: "female" },
    ];
    const [{ firstName }, , { gender }] = user;
    console.log(firstName, gender);

/*------ store particular object value in variable & print the value using created variable ------*/
    const user = [
        { userId: 1, firstName: "haresh", gender: "male" },
        { userId: 2, firstName: "rushabh", gender: "male" },
        { userId: 3, firstName: "komal", gender: "female" },
    ];
    const [{ firstName: item1 }, , { gender: item2 }] = user;
    console.log(item1, item2);
```

Operation Examples :

```
// ********** print all object key with pair value **********
    let obj = {
        name: 'parth',
        lname: 'savaliya',
        age: 22
    }

    let arr = []
    for (let i = 0; i < 5; i++) {    //for of loop print value
        arr.push(obj)
    }
    for (const iterator of arr) {
        for (const key in iterator) {   //for in loop print index
            console.log(key, ":", iterator[key])
```

```
    }
}

// ********** 5 time run loop & change particular index value **********
let obj = {
    name: 'parth',
    lname: 'savaliya',
    age: 22
}
let arr = []    // object store in array

for (let i = 0; i < 5; i++) {    //for loop print 5 time object
    // let newob = JSON.parse(JSON.stringify(obj))
    let newob = {...obj };
    // arr.push(obj)    //object push in array
    arr.push(newob)
}
for (const iterator of arr) {    //for of loop print value
    let index = arr.indexOf(iterator)
    if (index == 0) {
        iterator.name = "hareh"
        iterator.age = 555
        iterator.lname = "thummar "
    }
    for (const key in iterator) {     //for in loop print index
        console.log(key, iterator[key])
    }
}
```

*if else () :

```
//------- sort odd or even number using if else condition --------
for (let index = 0; index < 50; index++) {
    if(index % 2 === 0){
        console.log("even", index);
    }
    else{
        console.log("odd", index);
    }
```

```
}
```

* functions () :

- Arrow function used in callback function.
- function Expression used in objects or classes.

/*--- Function Declaration ---*/

- function ને **dry**(don't repeat yourself) પણ કહેવાય છે.
- don’t use **this** keyword in function. Because you don’t know this keyword scope. this by default target the function.

```
//------- function value store in variable --------
function returnValue(){
    return 2 + 5;
}
const a = returnValue()
console.log(a);                                  //7
```

```
//------- sum number using function parameter & argument --------
function sumTwoNumber(num1, num2){
    return num1 + num2;
}
const a = sumTwoNumber(5,10);
console.log(a);                                  //15
```

```
//------- sum number using prompt box in function --------
let a = +prompt();
let b = +prompt();

function sumInput(){
    return a + b;
}
let c = sumInput()
console.log(c);                             //add value in prompt box & get output
```

```
//------- check odd or even number using prompt box in function --------
let num = prompt();
function isEven(){
    if(num % 2 === 0){
        return true;
    }else{
        return false;
    }
}
console.log(isEven());                          //add value in prompt box & get output
```

```
//------ sort odd or even number in function -------
function isEven() {
    for (let index = 0; index <= 50; index++) {
        if (index % 2 === 0) {
            console.log("even", index);
        }
        else {
            console.log("odd", index);
        }
    }
}
isEven();                                       //print 1 to 50 odd & even number
```

/*--- Function Expression---*/

- એક variable માં function ને call કરાવીએ અને variable print કરીએ તેને **function expression** કહેવાય છે.
- આ function ને **anonymous(અનામી) function** પણ કહેવાય છે.

```
/*-------- function expression --------*/
const haresh = function() {
    console.log("happy birthday to you...");
}
haresh();                                               //happy birthday to you...
```

/*--- Arrow Function ---*/

- arrow function માં function keyword remove કરીને direct (=>) આવી રીતે લખીને arrow function create કરી શકાય છે.
- arrow function માં સામાન્ય રીતે એક input(parameter) હોય તો round bracket લેવાની જરૂર નથી તો વગર round bracket પણ ચાલે છે, પણ જો એકથી વધારે input હશે તો ફરજીયાત round bracket માં જ લખવું પડશે,

```
/*-------- arrow function --------*/
   const haresh = ()=> {
       console.log("happy birthday to you...");
   }
   haresh();                                          //happy birthday to you...

//------ shortcut arrow function --------
   const isEven = number => number % 2 ===0;
   console.log(isEven(4));                            //true

//------- sum number using function parameter & argument -------
   const sumTwoNumber = (num1, num2) => {
       return num1 + num2;
   }
   const a = sumTwoNumber(5,10);
   console.log(a);                                    //15
//--------- shortcut ----------
   const sumTwoNumber = (num1, num2) => num1 + num2;
   const a = sumTwoNumber(5, 10);
   console.log(a);                                    //15
```

Function hoisting :

```
/*------ declare the upper side function after call function ------*/
   hello();
   function hello() {
       console.log("hello haresh");
   }                                                  //hello haresh
```

Function inside function :

```
const app = () => {
      const myFunc = () => {
          console.log("this is first function");
      }
      const addTwoValue = (num1, num2) => {
          return num1 + num2;
      }
      const plusValue = (num1, num2) => num1 * num2;

      myFunc();
      console.log(addTwoValue(2, 5));
      console.log(plusValue(5, 5));
  }                                                    //this is first function
  app();                                               //2,5       //5,5
```

Lexical Scope :

- function ની અંદર child function બનાવેલ હોય તો એની અંદર આવતા variable ની value declair ના કરેલી હોય તો તેનો scope સીધો parent function આવી જશે. જો એ નામનો variable parent function માં declare કરેલો હશે તો તેને call કરશે અને અને જો ત્યાં declair કરેલો નહિ હોય તો તેનો scope સીધો outside માં global માં call કરાવવા જશે જો ત્યાં same નામનો variable મળી જશે તો તેને call કરી લેશે. આને function નો lexical scope કહેવાય છે.

```
/*--------- normal function lexical scope ---------*/
  const myVar = "normal function out scope call";       //1
  const app = () => {
      const myVar = "out scope call";                   //2
      function myFunc() {
          const myVar = "in scope call"                 //3
          console.log("inside myFunc :", myVar);
      }
      console.log(myVar);  //check the function scope
      myFunc();
  }
  app()

/*--------- function expression lexical scope --------*/
  const myVar = "function out scope call";     //1
  const app = () => {
```

```
        const myVar = "out scope call";          //2
        const myFunc = function() {
            const myVar = "in scope call"        //3
            console.log("inside myFunc :", myVar);
        }
        console.log(myVar);
        myFunc();
    }
    app();

/*--------- arrow function lexical scope --------*/
    const myVar = "function out scope call";   //1
    const app = () => {
        const myVar = "out scope call";        //2
        const myFunc = () => {
            const myVar = "in scope call"      //3
            console.log("inside myFunc :", myVar);
        }
        console.log(myVar);
        myFunc();
    }
    app();
```

*block scope **VS** function scope :*

- var is function scope.
- let and const are block scope.

```
/*--------- var scope --------*/
    {
        var haresh = "var scope";
    }
    console.log(haresh);            //var scope is global

/*--------- let scope --------*/
    {
        let haresh = "let scope";
```

```
        console.log(haresh);
    }
    console.log(haresh);              //Uncaught define ReferenceError

/*--------- const scope --------*/
    {
        const haresh = "const scope";
    }
    console.log(haresh);             //Uncaught define ReferenceError
```

Default parameter :

- જો સામાન્ય રીતે બે પરમેટર આવતા હોય તો આપણે **b parameter** ની **default value** આપણે **0 set** કરી દીધી છે પણ જો **default set** કરાવેલ ના હોય તો **error** આવે છે અને ફંકશન **work** નથી કરે. **set** કરાવેલ હશે તો **function** માં એક **argument** આપશો તો પણ ચાલશે. તો એ **a** ની જે **value** આવે છે તે **print** કરી દેશે.

```
    function addValue(a, b=0){
        return a + b;
    }
    let ans = addValue(4);
    console.log(ans);                // 4
```

Rest parameter :

```
     function myFunc(a, b, ...c){
        console.log("first value :", a);
        console.log("second value :", b);
        console.log("three value :", c);
    }
    myFunc(1,2,3,4,5,6,7);                  // 1,2,[3,4,5,6,7]
```

Example :

```
/*--------- all argument sum using function ---------*/
    function sum(...numbers) {
```

```
    let total = 0;
    for (let number of numbers) {
        total = total + number;
    }
    return total;
}
let ans = sum(1, 2, 3, 4, 5, 6);
console.log(ans);                    //output : 21
```

Parameter destructuring :

- parameter destructuring work with objects.
- most use in react.

```
const obj = {
    name: "haresh",
    gender: "male",
    age: 24,
}
function printObj({ name, gender, age }) {
    console.log(name, gender, age);
}
printObj(obj);                        //return object property value
```

Callback Function :

- આવા function ને higher order ફંકશન પણ કહેવાય છે.

```
function myFunc1() {                          //after run the function
    console.log("inside my func 1");
}
function myFunc2(callback) {                  //first run this function
    console.log("inside my func 2");
    callback();
}
myFunc2(myFunc1);

/*---------- callback function with two parameter ----------*/
function myFunc1(name) {
    console.log("inside my func 1");           //second run this function
    console.log(`your name is ${name}`);       //third run this function
```

```
}
function myFunc2(callback) {
    console.log("inside my func 2");            //first run this function
    callback("haresh");
}
myFunc2(myFunc1);
```

Callback Function :

- આવા function ને higher order ફંકશન પણ કહેવાય છે.

```
/*------------ function returning function ------------*/
function haresh() {
    function myFunc() {
        console.log("hello world");
    }
    return myFunc;
}
let ans = haresh();
ans();                                                  // hello world
/*------ short code ------*/
function haresh() {
    return function myFunc() {
        console.log("hello world");
    }
}
let ans = haresh();
ans();                                                  // hello world
```

* Array () :

```
/*---- first index * 2 multiplication in Array using function ----*/
const numbers = [4, 3, 5, 6];
```

```
function haresh(number, index) {
    console.log(`${number}*2 = ${number*2}`);
}
haresh(numbers[0]);
```

forEach Method :

- forEach method for loop ની જેમ જ work કરે છે. forEach method existing array return કરે છે.

```
/*---- first index * 2 multiplication in Array using function ----*/
const numbers = [4, 3, 5, 6];
function haresh(number, index) {
    console.log(`${number}*2 = ${number*2}`);
}
haresh(numbers[0]);

/*------- forEach funcUsing multiplication in array value -------*/
const numbers = [2, 5, 7, 9, 3];
numbers.forEach(function(number, index) {
    console.log(`index is ${index} number is ${number*2}`);
});

/*------- print first name in array of object using forEach -------*/
const hareshObj = [
    { firstName: "haresh", age: 20, gender: "male" },
    { firstName: "mohit", age: 21, gender: "male" },
    { firstName: "rutvik", age: 22, gender: "male" },
]
hareshObj.forEach(function(user) {
    console.log(user.firstName);
});



/*------ print first name in array of object using forEach with arrow function ------*/
const hareshObj = [
    { firstName: "haresh", age: 20, gender: "male" },
    { firstName: "mohit", age: 21, gender: "male" },
```

```
    { firstName: "rutvik", age: 22, gender: "male" },
]
hareshObj.forEach(x => {
    console.log(x.firstName);
});
```

output :

```
| haresh
| mohit
| rutvik
```

Map Method :

- map method હંમેશા એક new array return કરે છે. map method પણ for loop ની જેમ જ work કરે છે.

Example : 1

```
const number = [5, 2, 3, 8, 6, 4];
const square = function(number){
    return number * number;
}
const arrarValue = number.map(square);
console.log(arrarValue);                          //[25,4,9,64,36,16]
```

Shortcode :

```
const number = [5, 2, 3, 8, 6, 4];
const arrarValue = number.map(function(number){
    return number * number;
});
console.log(arrarValue);                          //[25,4,9,64,36,16]
```

Realistic Example :

```
const hareshObj = [
    { firstName: "haresh", age: 24, gender: "male" },
    { firstName: "mohit", age: 21, gender: "male" },
    { firstName: "rutvik", age: 22, gender: "male" },
    { firstName: "sweta", age: 23, gender: "female" },
];
const total = hareshObj.map((user)=>{
    return user.firstName + " " + user.age + " " + user.gender;
});
console.log(total);
```

```
| output :
| [haresh 24 male,
|  mohit 21 male,
|  rutvik 22 male,
|  sweta 23 female]
```

Filter Method :

- filter method એક existing array return કરે છે. map method પણ for loop ની જેમ જ work કરે છે. Return only true & false.

```
/*----- check even number -----*/
   const isEven = [5, 8, 3, 4, 9, 2, 1, 6];
   const checkEven = isEven.filter((number)=>{
       return number % 2 === 0;
   });
   console.log(checkEven);                                  // [5,3,9,1]

/*----- check odd number -----*/
   const isOdd = [5, 8, 3, 4, 9, 2, 1, 6];
   const checkOdd = isOdd.filter((number)=>{
       return number % 2 !== 0;
   });
   console.log(checkOdd);                                   // [8,4,2,6]
```

Reduce Method :

- reduce method એક existing array return કરે છે. reduce method પણ for loop ની જેમ જ work કરે છે.

```
/*------ sum of all number in array -------*/
   const user = [1, 2, 3, 4, 5, 6, 10];
   const total = user.reduce((accumulator, currentValue)=>{
       return accumulator + currentValue;
   });
   console.log(total);                          // 31

// accumulator    +    currentValue    +    return  (work flow)
    bydefault 1          bydefault 2            3
    3                         3                 6
    6                         4                 10
    10                        5                 15
    15                        6                 21
    21                        10                31

/*------ sum of all number using reduce in array of object -------*/
   const userCart = [
       {productId: 1, ProductName: "mobile", productPrice: 10000},
```

```
    {productId: 2, ProductName: "mobile", productPrice: 10000},
    {productId: 3, ProductName: "mobile", productPrice: 10000},
];
const totalAmount = userCart.reduce((totalPrice, currentProduct)=>{
    return totalPrice + currentProduct.productPrice;
}, 0);
console.log(totalAmount);

/*------ sum of all number using for loop in array of object -------*/
const userCart = [
    {productId: 1, ProductName: "mobile", productPrice: 10000},
    {productId: 2, ProductName: "mobile", productPrice: 10000},
    {productId: 3, ProductName: "mobile", productPrice: 10000},
];
let ft = 0;
for(let total of userCart){
    ft = total.productPrice + ft;
}
console.log(ft);
```

sort method :

- નાના limited data ને sort કરવા હોય તો આ method use થાય છે, જો વધારે data હોય તો JavaScript ની બીજી algorithm ની method use કરવી.
- sort method ascii code પ્રમાણે work કરે છે, array માં string માં કોઈ પણ નામ લખેલ હશે કે a થી z સુધી value હશે તો સીધી sort થઇ જશે પણ જો first later capital હશે તો તે first priority માં આવશે અને પછી small character વાળી value ને sort કરશે કારણ કે sort method ascii code પ્રમાણે work કરે છે. આમ પહેલા first, second, third આવી રીતે બધા character check કરશે પછી તેને sort કરશે પણ array માં number હશે તો આ પ્રમાણે work નહિ કરે.

```
/*----- ascending order -----*/
const numbers = [5, 2, 8, 400, 600, 200, 1400, 800, 950];
const sortNumber = numbers.sort((a, b) => a - b);
console.log(numbers);                    // [2, 5, 8, 200, 400, 600, 800, 950, 1400]

/*----- descending order -----*/
const numbers = [5, 2, 8, 400, 600, 200, 1400, 800, 950];
```

```
const sortNumber = numbers.sort((a, b) => b - a);
console.log(numbers);                    //[1400, 950, 800, 600, 400, 200, 8, 5, 2]
```

*** Realistic Example :**

```
/*------- sort product sorting lowToHigh & highToLow -------*/
const userCart = [
    { productId: 1, productName: "p1", productPrice: 500 },
    { productId: 2, productName: "p2", productPrice: 1200 },
    { productId: 3, productName: "p3", productPrice: 800 },
    { productId: 4, productName: "p4", productPrice: 200 },
    { productId: 5, productName: "p5", productPrice: 650 },
    { productId: 6, productName: "p6", productPrice: 2100 },
    { productId: 7, productName: "p7", productPrice: 50 },
];

const lowToHigh = userCart.slice().sort((a, b) => {
    return a.productPrice - b.productPrice;
})
console.log(lowToHigh);

const highToLow = userCart.slice().sort((a, b) => {
    return b.productPrice - a.productPrice;
})
console.log(highToLow);
```

find method :

- ખોળો limited data

```
/*---- find particular userId using arrow function in find method ----*/
const user = [
    { userId: 1, userName: "haresh" },
    { userId: 2, userName: "parth" },
    { userId: 3, userName: "rutvik" },
    { userId: 4, userName: "rushabh" },
    { userId: 5, userName: "divyesh" },
```

```
    { userId: 6, userName: "gunjan" },
];
const findName = user.find((item) => item.userId === 3);
console.log(findName.userId);                    // 3
console.log(findName);                           //{ userId: 3, userName: "rutvik" }
```

Every method :

- every method true કે false જ return કરે છે. every method માં બધી value check થશે જો બધી value true હશે તો statement true થશે નહીંતર false print થશે.

```
/*----- check even number in array -----*/
const a = [2, 4, 6, 8, 10];
const b = a.every((c) => c % 2 === 0);
console.log(b);                                  // true

/*----- check odd number in array -----*/
const a = [1, 3, 5, 7, 9];
const b = a.every((c) => c % 2 !== 0);
console.log(b);                                  // true
```

```
/*----- sum of all number using for loop in array of object -----*/
const userCart = [
    { productId: 1, ProductName: "mobile", productPrice: 10000 },
    { productId: 2, ProductName: "mobile", productPrice: 15000 },
    { productId: 3, ProductName: "mobile", productPrice: 21000 },
];
const checkPrice = userCart.every((cartItem) => cartItem.productPrice < 30000);
console.log(checkPrice);                         // true
```

some method :

- some method માં બધી value check થશે જો એક value true હશે તો statement true થશે નહીંતર false print થશે.

```
/*----- check odd number in array -----*/
const a = [1, 3, 5, 8, 9];
const b = a.some((c) => c % 2 === 0);
```

```
console.log(b);                         // true
```

```
/*--- check the value greater than 100000 using some method in array of object ---*/
const userCart = [
    { productId: 1, ProductName: "mobile", productPrice: 10000 },
    { productId: 2, ProductName: "mobile", productPrice: 15000 },
    { productId: 3, ProductName: "mobile", productPrice: 210000 },
];
const checkPrice = userCart.some((cartItem) => cartItem.productPrice > 100000);
console.log(checkPrice);
```

fill method :

- fill method માં તમે જે value જેટલી વાર print કરાવવા માંગો છો, તે fill method ની મદદથી થઇ શકે છે.
- આમ ત્રણ argument આવે છે (value, start, end) last end argument એ તમારે જ્યાં સુધી element select કર્યો હોય તે element કરતા એક element ઓછો select કરે છે. એટલે તમારે એક target કરતા એક element વધારે select કરવો પડે છે. આ method existing array માં જ work કરે છે.

```
/*----- print the value in your choice -----*/
const myArray = new Array(10).fill("haresh");
console.log(myArray);                           // [5,5,5,5,5,5,5,5,5,0]
```

```
/*----- custom value fill in your existing array -----*/
// argument (value, start, end)
const haresh = [1, 2, 3, 4, 5, 6, 7, 8, 9];
haresh.fill(0, 2, 5);
console.log(haresh);                            // [1, 2, 0, 0, 0, 6, 7, 8, 9]
```

* iterables () :

- જેમાં આપણે loop use કરી શકતા હોય અને એના ઉપર operation થઇ શકે એને iterable કહેવાય છે. (string, array) આ બધી type iterable છે. Object is not iterable.

```
/*----- string -----*/
    const haresh = "haresh";
    for (let item of haresh) {
        console.log(item);
    }

/*----- array -----*/
    const haresh = [1, 2, 3, 4, 5];
    for (let item of haresh) {
        console.log(item);
    }
```

* Array like object () :

- જેની પાસે length property છે અને જેને index થી access થઇ શકે એને array like object કહેવાય છે.

```
/*-------- string --------*/
    const firstName = "haresh";
    console.log(firstName.length);

/*-------- array --------*/
    const firstName = [1, 2, 3, 4, 5, 6, 7, 8, 9];
    console.log(firstName.length);
```

Map() set (key value pair) & get (key value) :

- map એ એક dictionary ની જેમ work કરે છે, તેમાં key અને value ની pair હોય છે, અને ખાલી key અને value જ store કરી શકીયે છીએ. object માં key ની type bydefault string જ રહે છે. પણ map માં તમે string, number, array, object કે symbol પણ રાખી શકીયે છીએ. map માં new keyword લખીયે તો તે એક empty dictionary બની જાય છે. map માં set property ની મદદથી key અને value store કરી શકાય છે, અને તેને print કરવા માટે get property નો use થાય છે. ખાલી key અને value pair આવતો હોય તો ઓ object ની જેમ map નો use કરી શકીયે છીએ. map is iterable. map એ key અને value ની pair output માં array માં print કરે છે.

```
/*------ set & get key value pair in map ------*/
    const person = new Map();                   // add empty map dictionary using new keyword
    person.set('firstName', 'haresh');          // add(set) string value in key
    person.set('age', 24);
```

```
person.set(123, 'user');                    // add(set) number value in key
person.set([1,2,3], 'onetwothree');         // add(set) array value in key
person.set({1:'one'}, 'onetwothree');       // add(set) object value in key

console.log(person);                        // print(get) all key & value
console.log(person.get('age'));             // print(get) particular key
console.log(person.get(123));
console.log(person.keys());                 // print only all keys

for(let item of person.keys()){             // print all property using for of loop
    console.log(item);
}
for(let item of person.keys()){           // print particular property using for of loop
    console.log(item, typeof item);
}
for(let item of person){                  // print key & value pair in array
    console.log(item);
}
for(let item of person){                  // print key & value pair in array
    console.log(item);
}
for(let [item, value] of person){                 // print key & value pair
    console.log(item, value);
}
```

/*------ set key value pair in array of array in map ------*/

```
const person = new Map([['firstName', 'haresh'], ['age', 24],[[1,2,3], 'onetwothree'], [123,
    'user']]);
console.log(person);                        //
```

/*------ clone object method-------*/

```
const obj = {
    key1: "value1",
    key2: "value2",
};
const obj2 = {...obj};
const obj2 = Object.assign({}, obj);
```

```
obj.key3 = "value3";
console.log(obj);
console.log(obj2);
```

* Optional chaining () :

- જો object ની property આપણે access કરતા હોય અને એ property object માં existing નહિ હોય તો output માં error આવી જાય છે. પણ જો error ના through કરાવવી હોય તો optional chaining નો use થઇ છે. simple dot ની આગળ question mark નો use કરવાથી તે output માં error ની જગ્યાએ undefined આવી જશે. એટલે કે આ property object માં exist નથી કરતી.

```
const user = {
    firstName: "haresh",
}
console.log(user.firstName);
console.log(user?.address?.houseNumber);
```

```
const haresh = {
    key1: "value1",
}
const user = haresh?.key2?.age;
console.log(user);
```

* Methods () :

- function inside an object.
- this keyword object માં બનાવેલા function માં લખવામાં આવે તો તે આખા object ની property ને access કરે છે. અને તમે object ની property ને access કરી શકો છો. અને object માં literals template use કરીને dynamic property પણ તમે use કરી શકો છો. object ની બહાર function બનાવીને object ની અંદર function declare કરાવીને this keyword ની મદદથી print કરી શકાય છે.

```
/*---- use this keyword & access object inside property dynamically ----*/
const user = {
    firstName: "haresh",
    age: 24,
    about: function(){
        console.log(`user name is ${this.firstName} & user age is ${this.age}`);
    }
```

```
    }
    user.about();

/*---- declare outside function & access outside function in multiple object ----*/
    function personInfo(){
        console.log(`person name is ${this.firstName} & age is ${this.age}`);
    }
    const user1 = {
        firstName: "haresh",
        age: 24,
        about: personInfo
    }
    const user2 = {
        firstName: "rutvik",
        age: 23,
        about: personInfo
    }
    const user3 = {
        firstName: "parth",
        age: 22,
        about: personInfo
    }
    user1.about();                          //person name is haresh & age is 24
    user2.about();                          //person name is rutvik & age is 23
    user3.about();                          //person name is parth & age is 22
```

- console માં ખાલી this keyword & window keyword pass કરવાથી window object (javascript ની property) મળે છે.

```
/*------ this keyword ------*/
    function myFunc(){
        console.log(this);
    }
    myFunc();                                   // print window object
```

- જો strict mode use કરીએ તો window object print નહિ થાય તો ખાલી undefined ની error આવી જશે. બધા developer એટલે strict mode use કરે છે.

```
/*------ use strict mode ------*/
    'use strict'
    function myFunc(){
        console.log(this);
```

```
}
myFunc();                                   // undefined
```

* function () :

- function ને call, apply, bind method use કરીને પણ call કરી શકાય છે.

1. Call () method :

```
/*-- call function using call method --*/
function user(){
    console.log("hello haresh");
}
user.call();                                    //hello haresh
```

- બીજા object ની value પહેલા object માં પાસ કરાવવી હોય તો call method થી get કરી શકીયે છીએ. simple રીતે પહેલો object print કરાવીને એમાં call method માં second object પાસ કરાવી દઈએ તો output માં second object ની value print થાય છે. તમારે parameter નો use કરવો હોય તો પણ use કરી શકીયે છીએ, જેમ નીચે આપેલા example પ્રમાણે થઈ શકે છે.

```
/*------ target second object value using first object with parameter using call method ------*/
const user1 = {
    firstName : "haresh",
    age : 24,
    about : function(hobby, music){
        console.log(this.firstName, this.age, hobby, music);
    },
}
const user2 = {
    firstName : "rutvik",
    age : 22,
}
user1.about.call(user2, "songs", "sonu");               // rutvik 22 songs sonu
```

```
/*------- outside declare function & access object using call method -------*/
   function about (hobby, music){
       console.log(this.firstName, this.age, hobby, music);
   }
   const user1 = {
       firstName : "haresh",
       age : 24,
   }
   const user2 = {
       firstName : "rutvik",
       age : 22,
   }
  about.call(user2, "songs", "sonu");                    // haresh 24 songs sonu
```

2. Apply () method :

- apply method માં square bracket માં function ની argument pass કરાવવી પડે છે.

```
/*---- target second object value using first object with parameter using apply method ----*/
   const user1 = {
       firstName : "haresh",
       age : 24,
       about : function(hobby, music){
           console.log(this.firstName, this.age, hobby, music);
       },
   }
   const user2 = {
       firstName : "rutvik",
       age : 22,
   }
   user1.about.apply(user2, ["songs", "sonu"]);                // haresh 24 songs sonu
```

3. Bind () method :

- bind method ને એક function માં store કરાવીને કે call કરી શકાય છે, બાકી બીજી method ની જેમ call નથી કરી શકાતી.

```
/*------- bind function using call method -------*/
   function user(){
```

```
    console.log("hello haresh");
}
const myFunc = user.bind();
myFunc();                                                    // hello haresh
```

```
/*----- target second object value using first object with parameter using bind method -----*/
const user1 = {
    firstName : "haresh",
    age : 24,
    about : function(hobby, music){
        console.log(this.firstName, this.age, hobby, music);
    },
}
const user2 = {
    firstName : "rutvik",
    age : 22,
}
const myfunc = user1.about.bind(user2, "songs", "sonu");
myfunc();                                                  // haresh 24 songs sonu
```

```
/*----- inside function call using bind method ------*/
const user1 = {
    firstName : "haresh",
    age : 24,
    about : function(){
        console.log(this.firstName, this.age);
    }
}
const myFunc = user1.about.bind(user1);
myFunc();                                                    // haresh 24
```

```
//----- shorthand code make Method in Object ------
const user1 = {
    firstName: "haresh",
    age: 20,
```

```
        about(){
            console.log(this.firstName, this.age);
        }
    }
    user1.about();                                          // haresh 20
```

* Arrow Function :

- arrow function ની અંદર this keyword નો use ના કરવો. જો this keyword નો use કરશો તો તે object print કરવાની ની જગ્યાએ window object print કરી દેશે

```
/*----- this keyword not use in arrow function inside object ------*/
// const user1 = {
//     firstName: "haresh",
//     age: 20,
//     about: () => {
//         console.log(this.firstName, this.age);
//     }
// }
// user1.about(user1);                                      // undefined undefined
```

***** OBJECT ORIENTED PROGRAMMING IN JAVASCRIPT *****

```
// function
// 1) that function create object
// 2) add key value pair
// 3) return of object
```

- first priority object 2 ની value ની રહેશે, જો object 2 માં same property હશે તો તે પહેલા print થશે એમાં નહિ હોય તો એ object 1 માં find કરશે એમાં હશે તો ત્યાંથી print કરી દેશે. બીજો object create કરીને એમાં object 1 pass કરો એટલે value get કરવા લાગે છે.

```
//**************** this is one more way to create empty object ****************//
    const obj1 = {
```

```
  key1: "value1",
  key2: "value2",
}
const obj2 = Object.create(obj1);
obj2.key2 = "newValue2";
obj2.key3 = "value3";
obj2.key4 = "value4";

console.log(obj2.key2);                    // newValue2
```

* proto & prototype chain :

- object 1 ની proto object 1 ને refer કરે છે. **JavaScript** માં **prototype** પણ એક અલગ **property** છે.

__proto__ , [[prototype]] ⇒ both are some

```
//***************** proto & prototype ****************//
  const obj1 = {
    key1: "value1",
    key2: "value2",
  }
  const obj2 = Object.create(obj1);
  obj2.key2 = "newValue2";
  obj2.key3 = "value3";
  obj2.key4 = "value4";
  console.log(obj2.__proto__);         // {key1: 'value1', key2: 'value2'}

//***************** proto & prototype chain ****************//
  const obj1 = {
    key1: "value1",
    key2: "value2",
    key3: [1,2,3,4],
  }
  const obj2 = Object.create(obj1);
  obj2.key2 = "newValue2";
  obj2.key3 = "value3";
  obj2.key4 = "value4";
  console.log(obj2.__proto__);         // {key1: 'value1', key2: 'value2', key3: array[4],}
```

```
//**************** proto chain Example ****************//
  const userMethod = {
    about: function(){
      return `${this.firstName} is ${this.age} year old`;
    },
    is18: function(){
      return this.age >= 18;
    },
    sing: function(){
      return "toon na na na la la...";
    }
  }
  function createUser(firstName, lastName, email, age, address){
    const user2 = Object.create(userMethod);
    user2.firstName = firstName;
    user2.lastName = lastName;
    user2.email = email;
    user2.age = age;
    user2.address = address;
    user2.about = userMethod.about;
    user2.is18 = userMethod.is18;
    user2.sing = userMethod.sing;
    return user2;
  }
  const getUser1 = createUser("rushabh", "patel", "rushabh@gmail.com", 18, "gujrat");
  const getUser2 = createUser("parth", "savaliya", "parth@gmail.com", 20, "mysore");
  const getUser3 = createUser("", "savaliya", "parth@gmail.com", 20, "mysore");

  console.log(getUser1);
  console.log(getUser2);
  console.log(getUser3);
  console.log(getUser1.about());              // rushabh is 18 year old
  console.log(getUser2.is18());               // true
  console.log(getUser3.sing());               // toon na na na la la...
```

Array Examples :-

```
/********** count first index & last index in array **********/
  let haresh = [10, 20, 30, 40, 50];
  let user = haresh[0] + haresh[haresh.length-1];              // 10 + 50
  console.log(user);                                          // 60

/********** check the first index & last index are same in array **********/
  let haresh = [10, 20, 30, 40, 50];
  if(haresh[0] === haresh[haresh.length-1]){
    console.log("element is same");
  }else{
    console.log("element is not same");
  }                                                           // element is same

/********** remove the all element last 4 number in array **********/
  let haresh = [];
  let cutele = [];
  for (let index = 1; index <= 100; index++) {                // loop 1 to 100
    haresh.push(index);
  }
  for (let i = 0; i < haresh.length; i++) {
    let element = haresh[i].toString();
    if(element[haresh[i].toString().length-1].includes("4")){
      haresh.splice(i,1)
      cutele.push(Number(element))
    }
  }
  console.log(haresh);          //print 1 to 100 numbers with removed no. last 4 no.
  console.log(cutele);          //removed no. list [4,14,24,34,44,54,64,74,84,94]
/*------ short hand ------*/
  haresh.forEach((x, i) => {
    if (x.toString()[x.toString().length-1].includes(4)) {
      haresh.splice(i, 1)
    }
```

```
})
console.log(haresh);          //print 1 to 100 numbers with removed no. last 4 no.
```

/******** add key value in object using if else condition *********/**

```
let obj = [
  {
    name: "parth",
    age: 22
  },
  {
    name: "haresh",
    age: 24
  },
  {
    name: "rushabh",
    age: 22
  },
  {
    name: "Ravat",
    age: 22
  },
];
let newfinal = obj.map(x => {
  if (x.age === 22) {
    x.education = "10th";
    return x;
  } else {
    x.education = "12th";
    return x;
  }
})
console.log(newfinal);
```

output :

```
// 0: {name: 'parth', age: 22, education: '10th'}
// 1: {name: 'haresh', age: 24, education: '12th'}
// 2: {name: 'rushabh', age: 22, education: '10th'}
// 3: {name: 'Ravat', age: 22, education: '10th'}
```

/***** all value addition in array *******/**

```
let arr1 = [1, 4, 6, 9, 5]
let plus = 0
```

```
  arr1.forEach(x => plus += x)
  console.log(plus);                // 26

/*----- short hand ------*/
  console.log(arr1.reduce((x,i)=>x+i));

/********** print center value in array *********/
  let arr1 = [0, 1, 1, 1, 0];
  let arr2 = [0, 2, 2, 2, 0];
  let arr3  = [...arr1.slice(1,4), ...arr2.slice(1,4)];
  console.log(arr3);                                    // [1,1,1,2,2,2]

/******** replace even number ********/
  let arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
  arr.forEach((x, i) => {
    if (x % 2 == 0) {
      arr[i]= "even";
    }
  })
  console.log(arr);             // [1, even, 3, even, 5, even, 7, even, 9, even]

/********* print object value using inside function ********/
  let obj = {
    name: "parth",
    age: 22,
    getfullname: () => {
     return  obj.name + " " + obj.age
    }
  }
  console.log(obj.getfullname());                   // parth 22
```

* JavaScript function ⇒ function + object Examples :-

- function એક free space ({}) આપે છે તેને prototype કહેવાય છે. prototype property" ખાલી function provide કરે છે.

- Name property → tells function name
- Function provide more useful property
- function provide call, apply, bind methods

```
/************ javascript function ===> function + object ************/

/*----- print simple function with object type -----*/

// function hello(){
//   console.log("hello world");
// }
// console.log(hello.name);                // hello

/*----- you can add your own properties -----*/

// function hello(){
//   console.log("hello world");
// }
// hello.myOwnProperty = "very unique value";
// console.log(hello.myOwnProperty);              // very unique value

/*---- check the prototype in function using if condition ----*/

// function hello(){
//   console.log("hello world");
// }
// if(hello.prototype){
//   console.log("prototype is present");
// }else{
//   console.log("prototype is not present");        // output
// }                                                   // prototype is present

/*---- check the prototype in object using if condition ----*/

// const hello = {key1: "value1"};
// if(hello.prototype){
//   console.log("prototype is present");
// }else{
//   console.log("prototype is not present");          // output
```

```
// }                                                    // prototype is not present

/*---- check the prototype in array using if condition ----*/
// const hello = ["value1", "value2"];
// if(hello.prototype){
//   console.log("prototype is present");
// }else{
//   console.log("prototype is not present");          // output
// }                                                    // prototype is not present
```

```
/*----- add property prototype in function ------*/
  function hello(){
      console.log("hello world");
  }
  hello.prototype.name = "haresh";
  hello.prototype.age = 20;
  hello.prototype.constructor = 20;
  hello.prototype.myFunc = function(){
      return "hello! how are you.."
  }
  console.log(hello.prototype);              // {name: "haresh", xyz: "xyz", constructure: f}
```

```
/*----- add methods & access method using prototype in function ------*/
  function createUser(firstName, lastName, email, age) {
     const user = Object.create(createUser.prototype);
     user.firstName = "haresh";
      user.lastName = "thumar";
      user.email = "haresh@gmail.com";
      user.age = 23;
      return user;
  }
  createUser.prototype.about = function () {
      return `${this.firstName} is ${this.age} year old`;
  };
```

```
createUser.prototype.about = function () {
    return this.age >= 18;
};
createUser.prototype.about = function () {
    return "toon na na na la la...";
};
const getUser1 = createUser();
console.log(getUser1);
console.log(getUser1.about());
```