(Nirof)

New class — [7th February-2023]

Java Script - class 1

Javascript Basics :-
↳ logic and functionality.

What is Javascript?

→ Light weight Programming Languages and Scripting Languages use to implements the Behaviours of the website.

History → Netscape navigator founded Javascript (1994)
Firstly it was called mocha, then Livescript then Javascript.

What can we do with JS :-

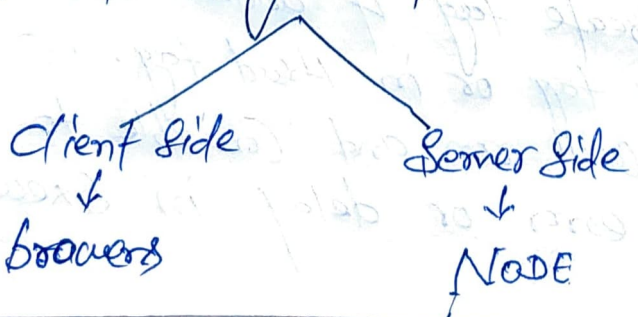
→ we can create web app/mobile application, Network apps.
→ CLI tools
→ Games.

(Created in 1995 (Netscape) - in 10 days developed "Scripting Language")

Client side Scripting languages Execute on web browers.
The JS Engine/Environment help to run JS code) in Chrome is called V8 - firbox - Spider monkey.
(DONT GO IN DEPTH :)

(Server Side). — To run Javascript outside the browers a C++ program added JS.
and NODE is Invented (by Ryan Dehl)

To Run Javascript



① What is Server?

→ A Computer which gives back data to client's Computer when client Searches Something.

→ To Run in Browser :-

→ Inspect in browser and go to 'Console' tab & then you can add :)

→ To Run in IDE :- (Code editor)

→ ① VS Code → Install

② Node.js → Install

Adding JS in Code

use <script> tag in HTML document.

Ex: <script> → to print or log.

Console.log("Namaste Duniya")

</script> ~ used for client side Scripting.

② we can add script tag inside.

→ Head tag

→ body tag.

Q) Best Practice?

→ Best Practice is to add in body tag below of all the HTML Codes add script in last Body tag.

why?

→ It will create lag if added above in the body tag or in Head tag, first script will run and Variable to parse that will error or delay in Execution.

Comment

* Comment in Javascript (//) (Ctrl+/ related :: //)
→ No significance in execution.

* External JS

Due to seperation of Concern, we will use external file for Javascript.
we create Javascript file use extension (.js)

* Linking <script src="index.js"></script>.

To run js file using Node :-

① VS Code → view (top bar)
② open terminal.
③ then make sure you are inside your working folder.
④ Command → Node index.js.

~~Int~~ Variables

Named memory location is called variable.

Creating variable in JS :- (var, let & Const)
As it is dynamic typed language we no need to tell which data type to use, it automatically defects from the value.

Ex :- let a = 5 (numbers)
let name = "Niraj" (string)
let status = true; (boolean)
let b = 12.5 (float)
↳ variable name.

(let keywords)

Var keywords

```
var a = 12;
var Name = "Niraj"
```

Let vs var

difference is of scope — Block, global.

→ let is a block scope variable.

```
{
  let a = 5; (only be used inside this block)
}
```

Eg:-

```
if (true)
{
  let a = 5;
}
Console.log(a); (error)
```

Now

→ var is a global scope variable (Can change/use in the code document)

→ let → redeclaration not allowed.

→ var → redeclaration is allowed.

Const variable → Fixed value of variable. Can't be changed.

```
Const a = 5
a = 6 (reassign not allowed)
```

No redeclaration.

Variable Naming :- Rules

① Cannot be a reserved keywords (let if x)
② Meaningful
③ Cannot start with number (16x)
④ No Space use '_'
⑤ CamelCase (FirstName)

Primitive types :- defined data types.

→ String – ("Niraj")
→ Numbers – (1, 2, 3, 4, 1.23, 1.54)
→ Boolean – true and false
→ Undefind – let a; not defined
→ null → empty variable (defined empty)

Note: you can see Prettier the code please used space my code.

Dynamic typing :- changing data type in js.

```
let a=5;
a = "Niraj";
Console.log(a); → Print (Niraj)
```

Reference types (datatypes)

① objects (multiple variable linked) variables)
② Arrays – (List of similar items js)
③ Functions.

① objects :- (top level entity for multiple linked variables)

```
let person = {
    firstName = 'Niraj'
    age = 22
    };
```

Properties.

To Access :-
- dot Notation (person.age)
- Bracket Natation (person['age'])

② Arrays :- ① used to Contain a list of items.

```
let names = ['Love', 'Niraj', 'Sangam'];
```

0 1 2 — Indexes.

To Access indexes

```
Names[1] - Niraj
names[0] → Love
names[3] ?
names[3] = 'Ramesh'; // value added.
```

names[1]; 2 // updation overwrite.

ECMA
- ECMA is a standard of javascript ECMA is an organizations which every each add updates in javascript.
- ES6 → (Launched in 2015)

operators
① Arithmetic - (+, -, *, /, %, **) → powers
② Assignment - (=, +=, -=, *=, /=, %=)
③ Comparision - (>, <, >=, <=, ===, !==)
④ Ternary (Condition) (Cond^{n} ? val1 : val2)
⑤ Logical (AND, OR, NOT)
⑥ Bitwise (Bitwise AND, Bitwise OR)

Pre/post → Increment/decrement operators.

(++x;) → pre-increment
↙ firstly increment the value
second, use the value

ex:- let x=10
Console.log(++x);
↓ 11

eg:- x++ → post increment operator
↘ first use the value second increment the value.

let a=6
Console.log(a++)
↓ 6

Assignment - x = x+5
also (x+=5)

- x = x+3
also (x*=3)

Comparision
↳ Answer will always be in True or false

=.

=== (strict equality) !== not equal

Equality

== (loose equality) === (strict equality) } == vs ===

== → Loose equality, value is same or not.

let num = 1
let str = '1' == gives true;

=== → Strict equality, value + data is same or not

let num = 1;
let str = '1' === gives false

Ternary operator :-

Cond^n ? val1 : val2

true → val1
false → val2

Ex :- age = 27
let status = (age >= 18) ? 'valid' : 'Cant';

@ Logical operator

And (Cond^n1 && Cond^n2 && Cond^n3)

if any condition is false the entire false All conditions have to be true.

OR (Cond^n1 || Cond^n2 || Cond^n3)

any condition is true then true and false, then only false.

Not !
true → false
false → true

② with Non Boolean (logical operators)

(true || false) → True

(true || true) → true

(false || false) → false

Now,

(false || 'Love') → Love

(true || 1 || 5) → 1

(Concept of falsy and truthy)

falsy
↓
undefined
null
0
false,
"
NaN

Truthy
↓
anything that is not false.
↓ truthy

(false || 'Love')

* Short Circuiting Concept in OR:

(false || 1 || 5)
↓
finds truthy then stop execution and prints (1)

* Bitwise operator :-

Bits → 0 (false)
Bits → 1 (true)

Bitwise AND → &

Bitwise OR → |

&

A	B	O/P
0	0	0
0	1	0
1	0	0
1	1	1

(1)

A	B	O/P.
0	0	0
0	1	1
1	0	1
1	1	1

② while loop

```
let i=0;
while (condition)
{ - - -
  i++; ← updation
}
```

③ DO-while loop :-

```
let i=0
do
{ - - -
  i++
} while(i<10);
```

This executes atleast one time condition is true or Not it executes one time atleast.

Note :-
To Run - open your terminal and put Command -
- pwd - check path
- node index.js

(Niraj Kumar Tiwari)

#4

JAVASCRIPT - CLASS - 2

Javascript Basic - 2 :-

multiple linked variable in single entity is 'OBJECT'

```
let a = {};  ← Empty object.
```

object has a key : value pairs.

```
Const rectangle = {
    length : 1,
    Breadth : 2;
}
```

(key) (value)

- Operator to access properties of object.

ex: let rectangle = {
    length : 2,
    breadth : 2,
}

Properties

function ← draw: function() { ← rectangle.draw() (to access)

Console.log("function Draw")

}

}.

[draw: function can also be written as draw()]

[It is called object oriented Programming.]

function for object creation.

1. factory function
2. Constructor function.

} two types of function.

1) function Create Rectangle() {

Create Rectangle();

}

let a = Create Rectangle();

[l, b, area]
a - length a.draw()
a - breadth for Create function.

let a = 5; → [5] - box
a

let a = { }
a is an object

[object] - Empty.

let a = { l : 1,
l : 2
};

[l → 1
l → 2] - object.
(a)

let a = {
Properties { l : 1
b : 2
draw() {
Console.log('drawing');
};

{l → 1, b → 2
draw() {} - object.

Input parameters for function.

function CreateRectangle(length, breadth) {
return rectangle = {
length,
breadth,
draw() {
log("draw"); } } }.

let rectangle obj1 = createRectangle(5, 4)

length, breadth

Value can change here.

(2) Constructor function -

We follow PascalCase Notation.

Camel → number of students

Pascal → firstword of word Always Capital.

```
function Rectangle() {
    this.length = 2;
    this.breadth = 3;
    this.draw = function() {
        alg("draw")
    }
}
```

(current object)

* Constructor function -

① defines the properties of method.

② does not returns.

New → keyword that return empty object.

let rectangleObj = new Rectangle();

we can also give Parameters here let change value.

```
function Rectangle(len, bre) {
    this.length = len;
    this.breadth = bre;
}
let rectangleObj = new Rectangle(5, 6)
a.draw()
```

(Object) (method)

```
draw()
{
    alg(this.length)
}
```

this is a so length will be printed.

Dynamic Nature of object:

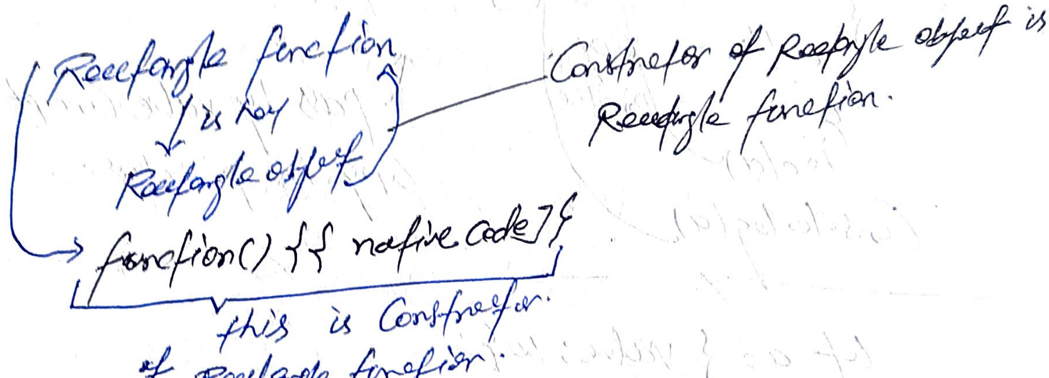
let
x -> 1   a.c = 4 ← addition   delete a.c;
b = 2
a;

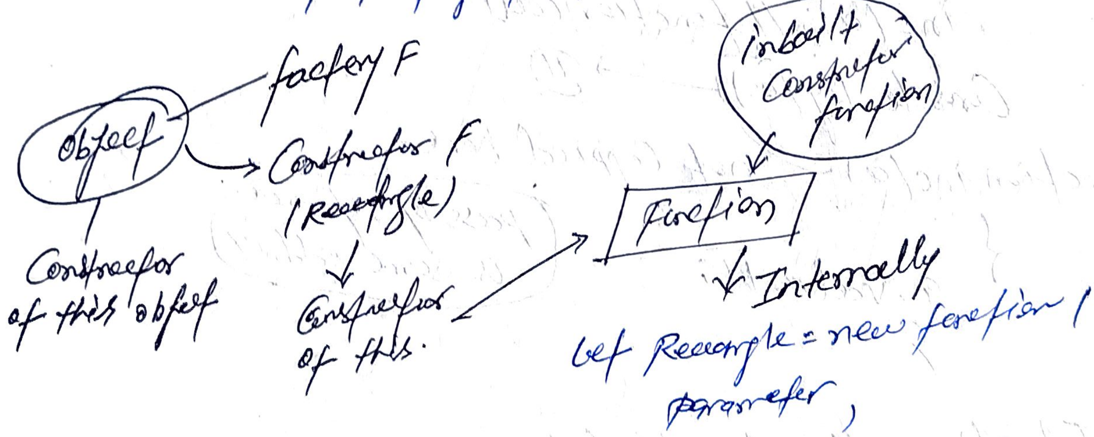
We can add or remove properties of object.

To add Rectangleobj.color = "yellow";
This will add color property in rectangle object.

To Remove
delete rectangleobj.color;

Constructor – function is also an object all object has a constructor.





function are objects.
as it have properties and entity.

difference Primitive type and Refrence type.

Primitive let a = 10 → a[10]
let b = a → b[10]
a++ → print(a) → 11
→ print(b) → 10

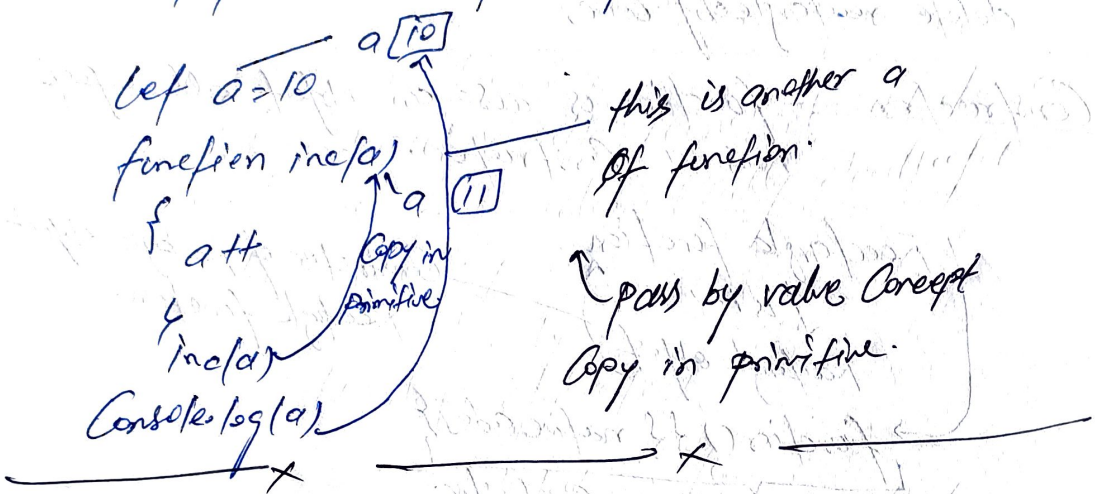
here Copy is created.

Reference

```
let a = {value: 10};
let b = a;
a.value++
Console.log(a); → 11
Console.log(b); → 11
```

In, object, address is passed so, both will represent the memorey location.

Note:- Primitive are Copied by this value
Reference are Copied by this address.




```
let a = { value: 10 };
inc(a); // functioncall
Console.log(a) → 11
function inc(a) { not Copied here.
  a.value++;
}
```

pass by reference is same address

Looping through objects:-

1. for-in loop.
2. for-of loop.

for-in loop

```
let room = {
  length: 2;
  breadth: 4;
};
```

```
for(let key in reef){
    console.log(key, reactangle[key])
}
```

to access key name

to access value of key.

for of → default works in objects

→ iterables → Arrays, maps.

for of in objects :- (HACK)

```
for(let key of object.keys(reef)){
    console.log(key)
}
```

use object.entries(reef) for value too.

we can use if else to know the property is present or not.

```
if('length' in reef){
    console.log("yes")
}else{ console.log("No")
}
```

object cloning (same to same one more)

- iteration
- Assign
- Spread

} rules.

① iteration

```
for(let key in Rectangle){
    console.log(key, Rectangle[key]);
}
```

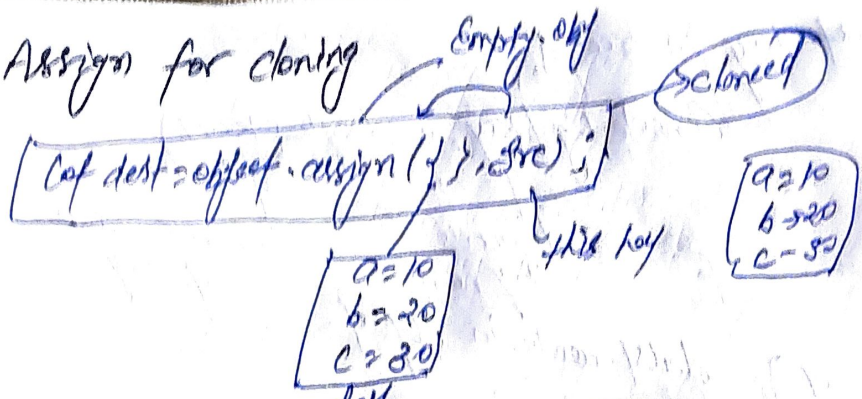
```
let obj2 = {};
```

we will copy all key and value of Rectangle in obj2 one by one. At first it will be empty.

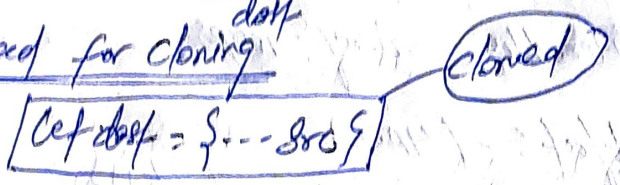
cloned

```
for(let key in src){
    dest[key] = src[key];
}
```

② Assign for cloning



③ Spread for cloning



~~Last~~ Garbage Collection :-

↳ find all variables / constants which are not in use and automatically deallocate their value.

↓
done by
(Garbage Collector)
↑
total in JS

↳ we have no control over Garbage Collector, runs in background itself.

Night

Javascript - class - 3 (22th february 2023.)

(In built objects && Arrays)

① Math object :- In built object for mathmatical functions.

→ Math.random ()
↳ generate Random no b/w 0 & 1

→ Math.max (2, 1, 4, 3)
↳ for maximum no.

→ Math.round (1.8)
↳ round of 1.8 ie 2.

→ Math.abs(-2)
↳ absolute, returns, positive of for positive and positive for negative here, -2 will be returned.

② String object :- There are two types of string in JS.

String → primitive
↳ let name = 'Niraj';

String Constructor functions
String → object
↳ let name = new String('name');
typeof(name) → object

we can convert primitive string to object using notation.

name.length, name.includes('Tree')
name.startsWith('tre'), name.endsWith('on');
name.toUpperCase(); name.toLowerCase();
name.min(), name.replace('tree', 'pw');
and multiple other functions.....

To split :-

let message = 'This is my message';
let word = message.split(" ");
Console.log(word);

③ Template Literal :-

to use single ' in string
① - Slash is used
these are the notations.

like
for new line (\n)
But another alternative without using \. slashy.
we use

Template Literal
↳ Back Tick is used
(``)

let msg = `This is
my : message`;
Same will be the output.

also we can add variables, in backtick using $
let msg = `This

```
let msg = 'This is my message.
Hello ${name}';
```
↑ In a same order and name = 'Niraj' will be printed

④ Date and time :-

Date :-

① `let date = new Date();`
`dg(date);` → current date and time.

② `let date2 = new Date('June 20 1998 07:20');`
`dg(date);`

③ `let date3 = new Date(1998, 5, 20, 6);` → Time (hr)
↑ year ↑ month (indexing starts from 0 is Feb) ↑ date

also
`date3.getFullYear()` ← `date3.setFullYear(1944)`
`dg(date3);`

Function method (Getter/Setter)

when we use function/method to
Set value → Setter
Get value → getter

Arrays :- (Object / Reference type Collection of all types of item)

- Adding new elements
- finding elements
- Removing elements
- Splitting Elements
- Combining Elements.

① Creation let number = [1, 8, 5, 7] 'Tiwari';

② Adding / pushing new elements :-
we can access array using Indexes
number[0] → 1 (value in index 0)

Insert :-
- end
- Begining
- Middle

let number = [1, 4, 5, 7]

(i) End → [1, 4, 5, 7, 9]
→ number.push(9)

(ii) Beginning → [8, 1, 4, 5, 7]
↳ numbers.unshift(8)

(iii) middle → number.splice(2, 0, 'a', 'b', 'c', 'd']
index, deletion, adding

② find out numbers (Searching Element)
numbers.indexOf(2);
→ if we want to check if a number exit in an array.

```
if (numbers.indexOf(10) != -1)
  Console.log('present')
```
↳ Not right way.

Used prefics

```
Console.log(numbers.include(7));
```
↳ true/false (Modern)

Adv numbers.indexOf(4, 2);
↓ ↓
search index to → (-1) if
start

(-1) is answer when your desire index which is not present.

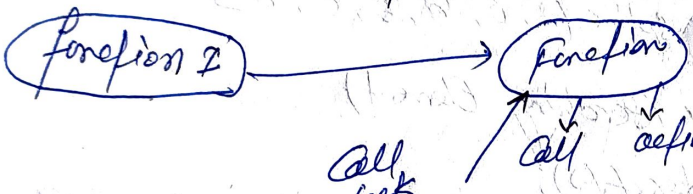
* we have done these on primitive
Now on Reference

```
let Courses = [
    { no: 1, names: 'Love' },
    { no: 2, name: 'Babbar' } ]
log(Courses);
```

↳ Array of object is created.

In reference we can't find using indexof and includes because searching in reference is not same as primitive.
for primitive it search by value.
for Reference it search by address.
we use Callback function. here

↓
function passed into another function as argument, which is then invoked inside the outer function to complete action




```
let c = Courses.find(function(courses) {
    return courses.name === 'Love' });
log(c);
```

Syntax:-
arrayName.find ← mediate function i.e condition to find object)

predicate object. — function(course) {
return course.name === 'Love' };

Arrow function (many Concise)

- let Course = Course.find(Course => Course.name === 'Love');

```
we rename {
    return
}
```

only when we have single value.

No input parameter then arrow function

④ Removing Element:- [1,2,3,4]

- end → pop()
- Beginning → shift()
- middle → splice(3, 1) — no of elements you want to delete.
  - main

⑤ Emptying an Array:-

number=[1,2,3,4,5] — automatically removed by garbage collector then

① number=[] — it's not deleted still it's stored.

for deleting.

② number.length=0 — this is what we do to make any empty.

③ also

number.splice(0, numbers.length)
index + no of element you want to delete)

④ also, while (numbers.length > 0) numbers.pop() — using loop.

⑥ Combining and Slicing Arrays:-

```
let first=[1,2,3];
let second=[4,5,6];
using Concat() method
let Combined=first.Concat(second);
```

Slicing, using slice() method.

[1, 2, 3, 4, 5, 6]

↓ we this

let courses = [

slice (,) → (x, y)

start Index — end Index

x → range where x is included

y is excluded.

[1, 2, 3, 4, 5, 6]

0 ① ② ③ ④ ⑤

Index

↓ Exclude

slice(2, 4) to get (3, 4) - slice

* if we give one parameter.

slice(2)

↳ from 2nd index all removed.

* if slice()

↳ Copy of original array.

called as slicing

* Spread operator :-

let first = [1, 2, 3]

let second = [4, 5, 6]

let combined = [...first, ...second]

also to add

let combined = [...first, '0', ...second, '6']

to copy

let another = [...combined]

Iterating on Array :-

loop ↳ for of loop is on iterables.

↳ (foreach) - also

let array = [1, 2, 3, 4]

for of [for (let value of array) {
console.log(value)
}

for each [array.foreach(function(numbers) {
console.log(numbers)
});

[DO change this to arrow function ..

Joining Array — num = [1,2,3]
to join them like (1,2,3)
using join() method.
num = [1,2,3,4]
const joined = num.join()
cg(joined) → (1,2,3,4)

Split() method.
Created an array.
let msg = "this is my message";
let parts = msg.split(" ")
cg(parts)
['This', 'is', 'my', 'message']
let joined = parts.('-')
cg(joined)

* ~~If~~ Sorting Arrays:- using sort() method.
~~Let~~ Sort is to arrays in increasing or decreasing order.
by default assending order.

let num = [10, 50, 20, 60, 30]
num.sort()
cg(num)
→ [10, 20, 30, 40, 50, 60]
also reverse using.
num.reverse()
[60, 50, 30, 20, 10]

→ we can't do sort() in object like this we have to add predicate function.

(*) Filtering Arrays:-
↳ using filter();
number.filter( )
↳ callback function.

```
let number = 2
let num = [1, 2, -3, -4]
let filtered = num.filter(function(value) {
    return value >= 0
});
cl(filtered) [1, 2] expected.
```

Mapping Arrays

map each element of array of something else, some like list in ZL

map() method

```
let numbers = [7, 8, 9, 10];
number.map(function(value) {
    return 'student.no' + value;
});
```

['student.no7', 'student.no8', ...]

mapping with object

```
let num = [1, 2, -3, -5]
let filtered = num.filter(value => value >= 0)
    cl(filtered)
let item = filtered.map(function(num) {
    let obj = { value: num };
    return obj;
});
    cl(item)
```

[{value: 1}, {value: 2}]

Javascript - class - 4 (Basic-4)

function :- A block of code that fulfills a specific task.

syntax :-

keyword → function, functionname → printCounting

```
① function printCounting () {
     Console.log ("Counting") }
```

] function body.

* why functions?
  ↳ Reusability to reduce Bulky Codes
  Remove / Reduce Bugs.

function Declaration :-

I/P parameters

```
(1) function run() {
      Console.log("running")
    }
```

To Call → run()

Hoisting in Javascript is Concept where, process of moving function declaration to the top of file. done by JS engine.
↳ by the help of this we can call function anywhere.
Only works for function Declaration.

(9.) function Assignment
↳ giving variable to a function.
(assignment) (assigning)

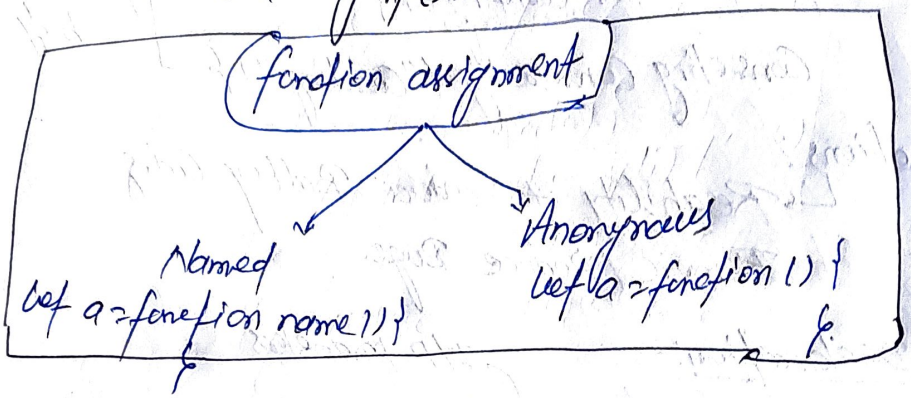
```
let stand = function walk () {
     Console.log ("walking"); }
```

To Call → stand () Not walk();

Hoisting Dosn't works here
↳ only for function declaration.

② Anonymous → Name of function not present.

```
let jump = function () {
    console.log("walking") }
call -> jump();
```



Dynamic functions:-

```
function sum(a, b) {
    return a+b; }
```

1) cg(sum()); // will give Nan (undefined for b)
2) cg(sum()); // will give Nan (undefined for a & b)
3) cg(sum(1,2,3,4,5)) // only 1 & 2 will be taken rest will be waste ans - ③

Stored in Argument object in fs.

Special object - Arguments (for multiple passing of argument)

```
let sum(a,b) {
    let total = 0
    for (let value of arguments)
    total = total + value;
    return total; }
let ans = sum(1,2,3,4,5)   we can increase this to get new value
console.log(ans)
```

Rest operator:- we can handle multiple parameters in function using Rest operator

This will create Array.

```
function Sum(...args) {
    cfg(...args)
}
Sum(1,2,3,4,5,6)
  ↳ [1,2,3,4,5,6]
  will be stored in array.
```

②
```
function Sum(num, value, ...args) {
    cfg(...cfgs); }
Sum(1,2,3,4,5,6);
```

1 stored in num
2 stored in value
Rest operator stored in arguments (..cfgs) It is a last parameter after this No parameters is allowed.

```
X(...args, num)   Not allowed X
```

Default parameters? - default default

```
function insert(P, r=10, y=2) {
    between p*r*y//100; }
```
all rest have to be default.

```
cfg(interest(1000,5));
```
↳ will give 100 taking y=2 default.

if user gives input then input will be taken if not default will be taken.

```
let person = {
    fname: 'Niroj';
    lName: 'Kinder tiwari'; };
function fullname() {
return `${person.fname} ${person.lname}`;
}
cfg(fullname());
```
This is only read only function.

to manipulates:

This will creafe Array.

```
function sum(...args) {
    clg(...args);
}
sm(1,2,3,4,5,6)
  ↳ [1,2,3,4,5,6]
  will be stored in array.
```

②
```
function sum(num, value, ...args) {
    clg(...args); }
sum(1,2,3,4,5,6);
```

Stored in num — Stored in value — Rest operator stored in arguments (...args) ↳ It is a last parameter after this No parameters is allowed.

X(...args, num) ← Not allowed X

Default parameters :-

```
function insert(P, r=10, y=2) {
    between p*r**y/100; }
```
(all rest have to be default)

clg(insert(1000,5)); ↳ will give 100 taking y=2 default.

if user gives input then input will be taken if not default will be taken.

```
let person = {
    fname: 'Niraj';
    lName: 'Kumar tiwari'; };
function fullname() {
return `${person.fname} ${person.lname}`; }
clg(fullname());
```
↑ This is only read only function to manipulates!

Getter, Setter :-

```
let person = {
    fname = 'Love'
    lname = 'Babbar',
  get fullName () {
      Return ` ${person.fname} ${person.fname};
  }
  // get fullName (value) {
        let parts = value.split (' ');
      this.fName = parts[0];
      This.lName = parts[1];
Next line  }  };
```

To Call :-

```
clg (person.fullNames);
        (getter) — getters
person.fullName = 'Niraj Kumar Tiwari';
clg (person.fullName); ← setters.
```

ERROR HANDLING :-

Using Try & Catch block.

```
Try {
  Code. if error goes to Catch.
  } Catch () {
        // Custom error message.
      }

let person 1 = {
      fname: 'Love';
      lName: 'Babbar';
    get fullName () {
      return ` ${person1.fname}
          ${person1.lname}`;
      },
    • Set fullName (value) {
```

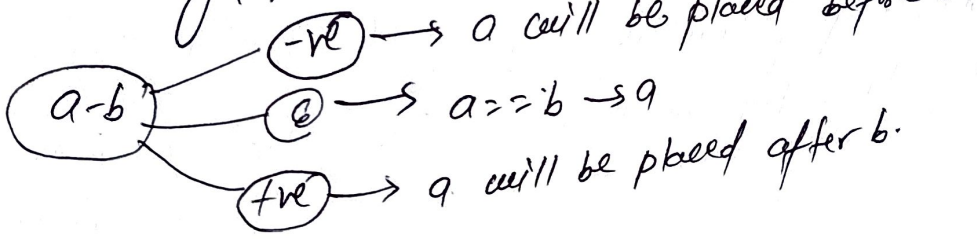
```
if typeof value !== string ) {
throw new Error
let parts = value.split(' ');
this.frame = parts[0];
this.name = parts[1]; } ;
```

```
try {
person1.function = true;
}
catch(e) {
alert(e);
}
```

Scope :- ↳ Lifetime or Lifely Lifespan of variable is Scope.

```
{ let a=5;
{ alg(a); → error
```
Block scope

Sorting :-
```
let a = [10, 30, 50, 60, 20, 80]
a.sort(function (a,b) {
    return a-b;
});
alg(a);
```
for assending
(b-a) for discading.



Redcing an Array :-

using reduce method:

```
let arr = [1, 2, 3, 4];
let total = 0,
for(let value in arr)
total = total + value;
Console.log(total);
```

// to reduce we write call back function using 2 parameters,

accumulator (total) — currentValue (loop)

```
let total sum = arr.reduce((accumulator, currentValue) => accumulator + currentValue, 0);
```

↑ accumulator initialized to 0.

```
Console.log(total sum);
```

→ (10)

Working :-

[1, 2, 3, 4]

accumulator = 0

current value = 1

accumulator = 0 + 1 = 1

current = 2

Accumulator = 1 + 2 = 3

current = 3

current = 4

accumulator = 6 + 4 = (10)

(3+3)