

JavaScript - class 1

JavaScript Basics :-

↳ Logic and Functionality.

What is JavaScript?

→ Light weight programming language & Scripting language use to implement the Behaviour 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 app / network apps
- CLI tools
- Games

Client Side Scripting language executes on web browser.

The JS Engine (environment help to run JS code) in Chrome is called V8

Firefox → Spider Monkey

DONT GO
IN DEPTH
😊

Server side,
To run JavaScript outside the Browser
a C++ program added with JS
and NODE is invented (by Ryan Dahl)

To run JavaScript

Client side



Browser

Server side



NODE

Q) What is Server?

→ A computer which gives back data to client's computer when client searches something.

→ To Run in Browser :-

→ 'Inspect' in browser & go to 'Console' & then you can code :-

→ To Run in IDE :- (Code editor)

- 1) VS code → Install
- 2) Node.js → Install

Adding JS in Code

use `<Script>` tag in HTML document.

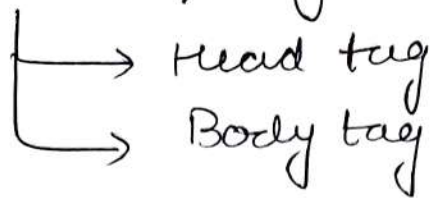
ex:- `<Script>` → to print or do work.

`console.log("Namaste Duniya");`

`</Script>`

↑
used for
client side
Scripting

✓ We can add script tag inside



Q) Best practice?

→ Best practice is to add in Body tag below of all the HTML codes. add Script in last of Body tag.

Why?

→ It will create Bug if added above in the body tag or in Head tag, First script will run & can't able to parse that will cause error or Delay in execution.

✓ Comment in Javascript using Forward slashes (//)
→ No significance in execution.

External JS

Due to Separation of concern, we will use external File for Javascript.

we create javascript File we 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.

Imp

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 detects from the value.

Ex:- let a = 5 (~~int~~) (number)

let name = "Tunwesh" (String)

let status = true; (boolean)

let b = 12.5 (Float)

↑
Variable
name

Let
keyword

Var keyword

var a = 12,

var name = "Tunwesh"

Let v/s 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
(anywhere 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 (update not
allowed)

No redeclaration

Variable Naming

Rules

- ↳ cannot be a reserved keyword (let if x)
- ↳ meaningful
- ↳ cannot start with number (1bX)
- ↳ no space use '_'
- ↳ camelCase (firstName)

primitive Types

→ defined data types

- String → ("Turwash")
- Number → (1, 2, 3, 4, 1.23, 5.64)
- Boolean → true or false.
- Undefined → (let a;) not defined
- null → empty variable (defined empty)

Dynamic typing.

↳ changing data type in JS

```
let a = 5;
```

```
a = 'Turwash';
```

```
console.log(a);
```

↳ Turwash
printed

Reference Types (datatypes)

- (1) Objects (multiple variables linked)
- (2) Arrays (list of similar ^{items (js)} ~~datatypes~~)
- (3) Functions

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

✓ let person = {

 firstName = 'Turvash',

 age = 24

};

properties

To Access :-

└ dot Notation (person.age)

└ Bracket Notation (person['age'])

② Arrays :-

↳ used to contain a list of items

✓ let names = ['love', 'rahul', 'Sangram'];

To Access :-

↓
indexes

↓ ↓ ↓
0 1 2
└──────────────────┘
Indexes.

names[1] → rahul

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 organisation which every year add updates in JavaScript.
ES6 → launched in 2015

Operators

- (1) Arithmetic (+, -, *, /, %, **, ^(power))
- (2) Assignment (=, +=, -=, *=, /=, %=)
- (3) Comparison (>, <, >=, <=, ===, !==)
- (4) Ternary (condition) (condⁿ ? val1 : val2)
- (5) ~~Bitwise~~ 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:- let a = 6 x++ → post Increment Operator

First use the value
Second increment the value

console.log(a++)

↓
6

Assignment

✓ $x = x + 5$
also

$x += 5$

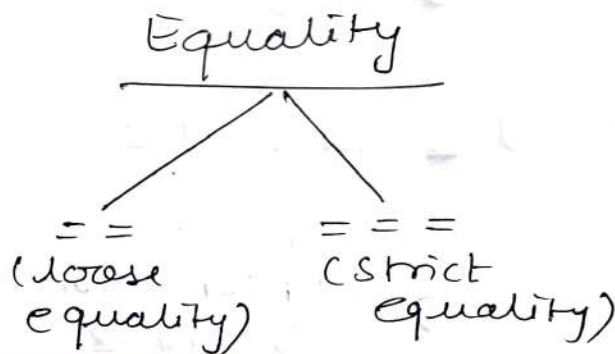
✓ $x = x * 3$
also

$x *= 3$

Comparison

✓ answer will always be in True or False.

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



$==$ v/s $===$

$==$ → loose equality, value is same or not

let num = 1

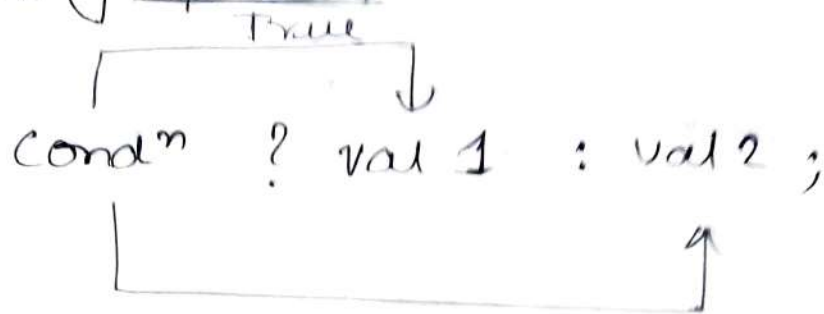
let str = '1' $==$ gives true;

$===$ → strict equality, value + data is same or not

let num = 1

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

Ternary Operator :-



ex:- age = 27

let status = (age >= 18) ? 'vote' : 'cant';

Logical Operator

And

(condⁿ1 && condⁿ2 && condⁿ3)



if any condition is False
the entire False

All conditions have to be true

OR

(condⁿ1 || condⁿ2 || condⁿ3)



any condition is true
then True

all False then only
False.

NOT !

True → False

False → True

With Non Booleans (Logical Operator)

(true || false) → true

(true || true) → true

(false || false) → false

Now,

(false || 'love') → love

(true || 1 || 5) → 1

Concept of Falsy & Truthy:

Falsy
↓
undefined
null
0
false
NaN

Truthy
↓
anything that is
not Falsy

truthy ✓
↓

(false || 'love')

* Short Circuiting Concept in OR

(false || 1 || 5)

↓

Finds truthy
then stop
execution
and prints (1)

Bitwise Operator

Bits \rightarrow 0 (False)
 \rightarrow 1 (True)

Bitwise AND \rightarrow &

Bitwise OR \rightarrow |

&

| A | B | O/P |
|---|---|-----|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

|

| A | B | O/P |
|---|---|-----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Operator precedence

let $c = a + b * d / c$;
which operator first?

use brackets to resolve problem of precedence.

let $c = (a + (b * d) / c)$

Control Statements :-

- 1) If-else
 - 2) Switch
- } two ways.

① if-else :- (if-elseif-else)

single { if (condition) {
 — —

{ else if (condition) }

can be multiple

multiple }
Single } else }
 } }

② Switch-case :-

logic $\left\{ \begin{array}{l} \text{input } 1 \rightarrow A \\ \quad \quad 2 \rightarrow B \\ \quad \quad 3 \rightarrow C \end{array} \right.$

Syntax Switch Case :-

Switch (expression) {

Case 1 : — —
break;

Case 2 : — —
break;

default : — —

}

break

after executing
the case

breaks the
Control Statement
& will not
execute further

Loops :- (Repetition of task)

- 1) For loop
- 2) while loop
- 3) Do-while loop
- 4) what is an infinite loop?
- 5) for-in loop
- 6) For-of loop

① For loop

```
for (let i=0 ; i<5 ; i++ )  
    {  
        console.log(i);  
    }
```

↑
initialisation

↑
condition

↑
updatation

0 1 2 3 4

2) while loop \leftarrow initialization
 $\text{let } i = 0;$
 while (condition)
 {
 — — —
 } $i++;$ \leftarrow updation.

3) Do-while loop :-
 $\text{let } i = 0$
 do
 {
 — — —
 } $i++;$
 while ($i < 10$);

This executes atleast one Time
condition is True or Not it executes
one Time atleast.

JavaScript - Class 2

Javascript Basics-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

ex: • operator to access properties of Object

let rectangle = {

properties ← { length: 2,
breadth: 4,

function ← draw: function() {

console.log("Function Draw")

}

}

rectangle.draw()(
(to access)

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

Function for object Creation

+ factory function
+ construction function } two types of creation.

Factory Function

1) Function createRectangle() {

(place your object code here)

return rectangle; ← write return at last by writing object name.

OR

2) Function createRectangle() {

return rectangle = {
- - -
}; ← Fixed values

write return here

Calling Factory Function :-

```
let name = createRectangle();  
           ↓           ↑  
       object      returns object  
       stored here  
  
console.log(name);
```

↑
to show the function which prints object.

Input parameters for Function.

function createRectangle (length, breadth) {

return {rectangle = {

length,

breadth,

draw() {

clog("draw");

}

let rectangleObj1 = createRectangle(5, 4)

value
can
change
here

length

breadth.

② Constructor Function

↳ we follow pascal Notation. ✓

camel → numberObjStudents

✓ pascal → FirstWordOfWordAlwaysCapital

function Rectangle() {

this.length = 2;

this.breadth = 3;

this.draw = function() {

alert("draw")

}

↑
current
object

* Constructor function

- defines the properties of methods
- does not return

new → keyword that returns empty object.

let rectangleObj = new Rectangle();

↑
we can
also give
parameters
here to
change value.

function Rectangle(len, bre) {

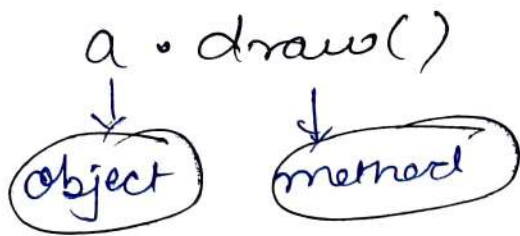
this.length = len;

this.breadth = bre;

}

}

let rectangleObj = new Rectangle(5, 6)



draw()
{
 csg (this.length)
}

this is or so a length
will be printed.

Dynamic Nature of object

↳ we can add, or remove
property of object.

To add ~~let a = 5;~~

rectangleObj.color = "yellow";

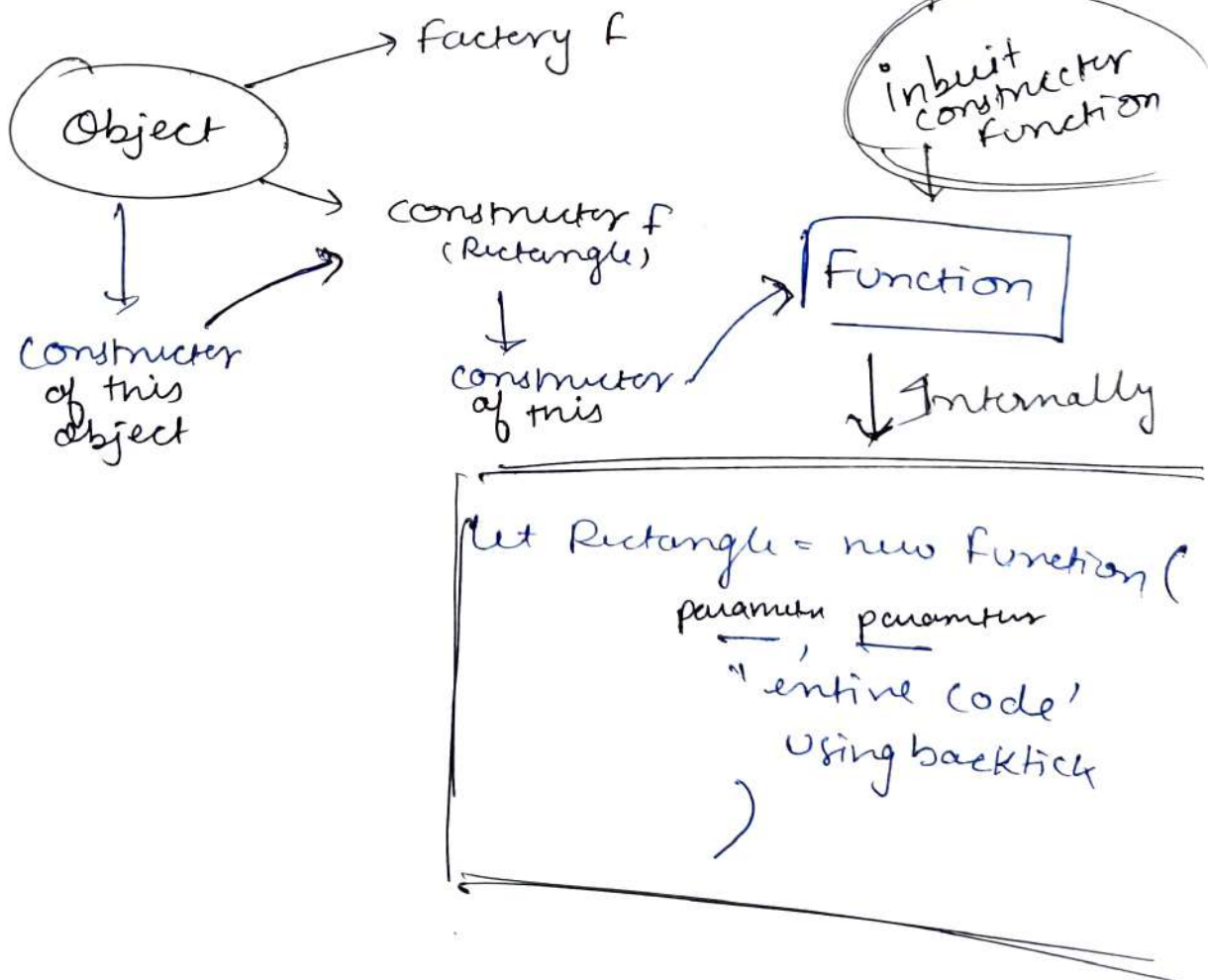
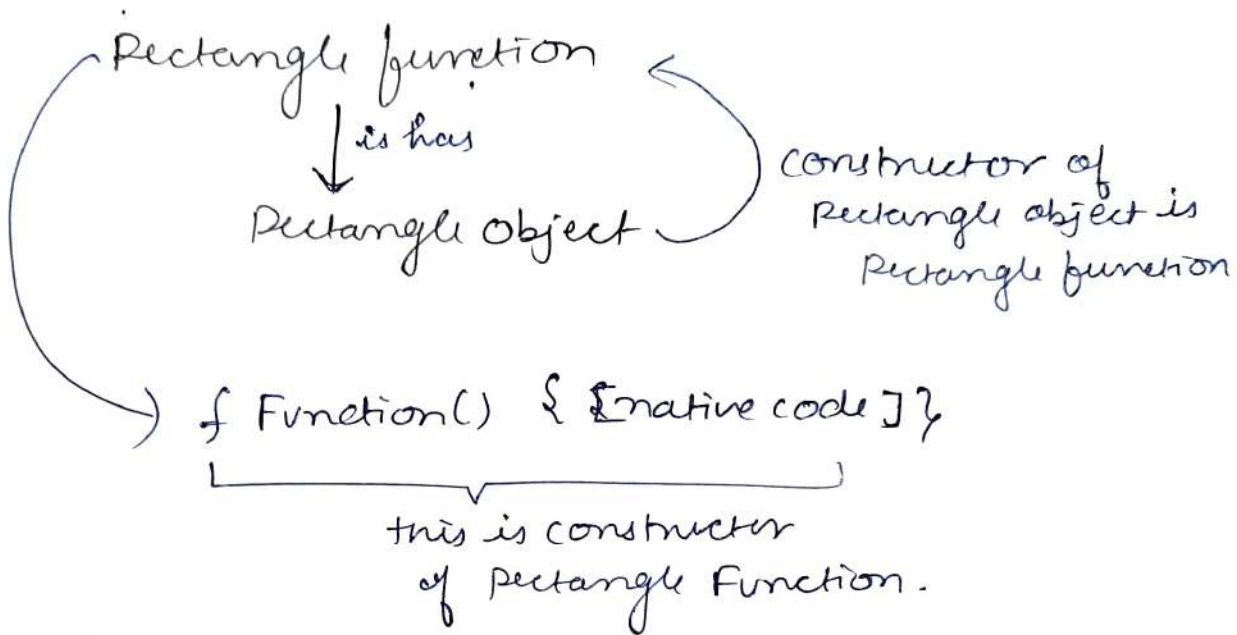
↓
This will
add
color
property
in
rectangle
object.

To remove

delete rectangleObj.color;

CONSTRUCTOR

Function is also an object
all object has a constructor



✓ Functions are Objects

as it have
properties & entity ✓

Difference primitive type & Reference 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++

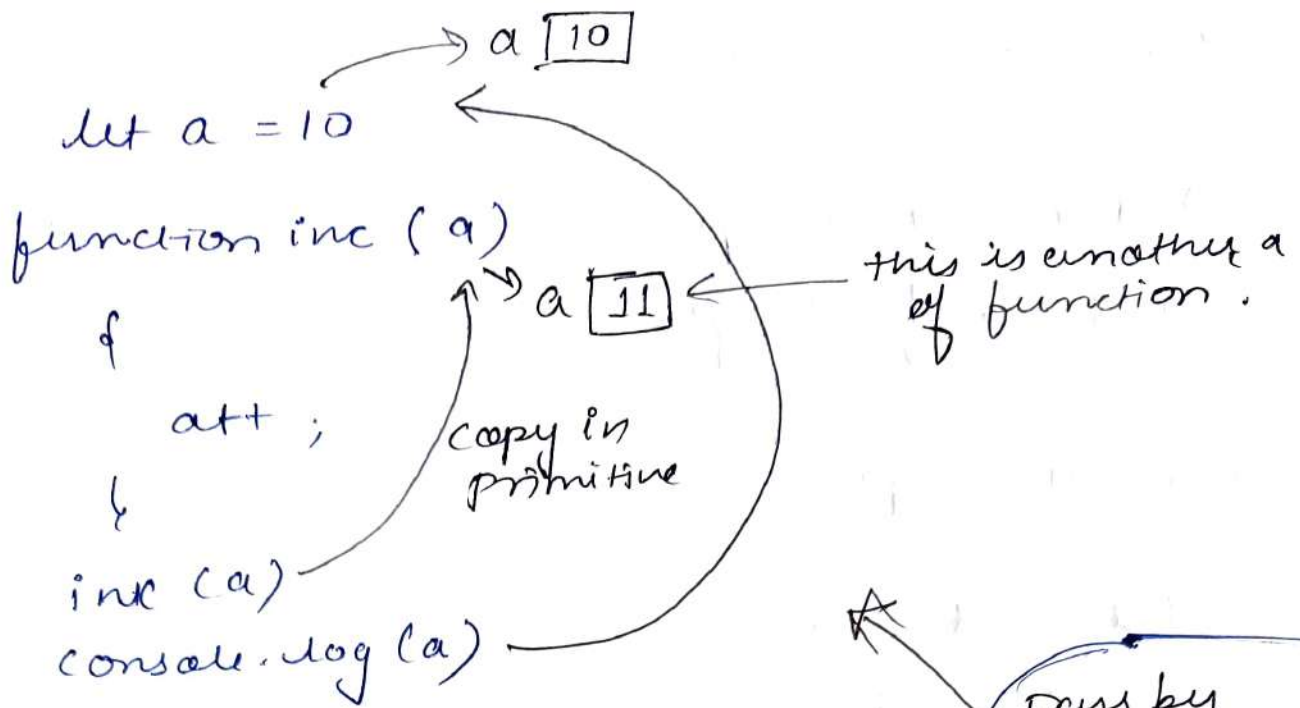
console.log(a); → 11

console.log(b); → 10

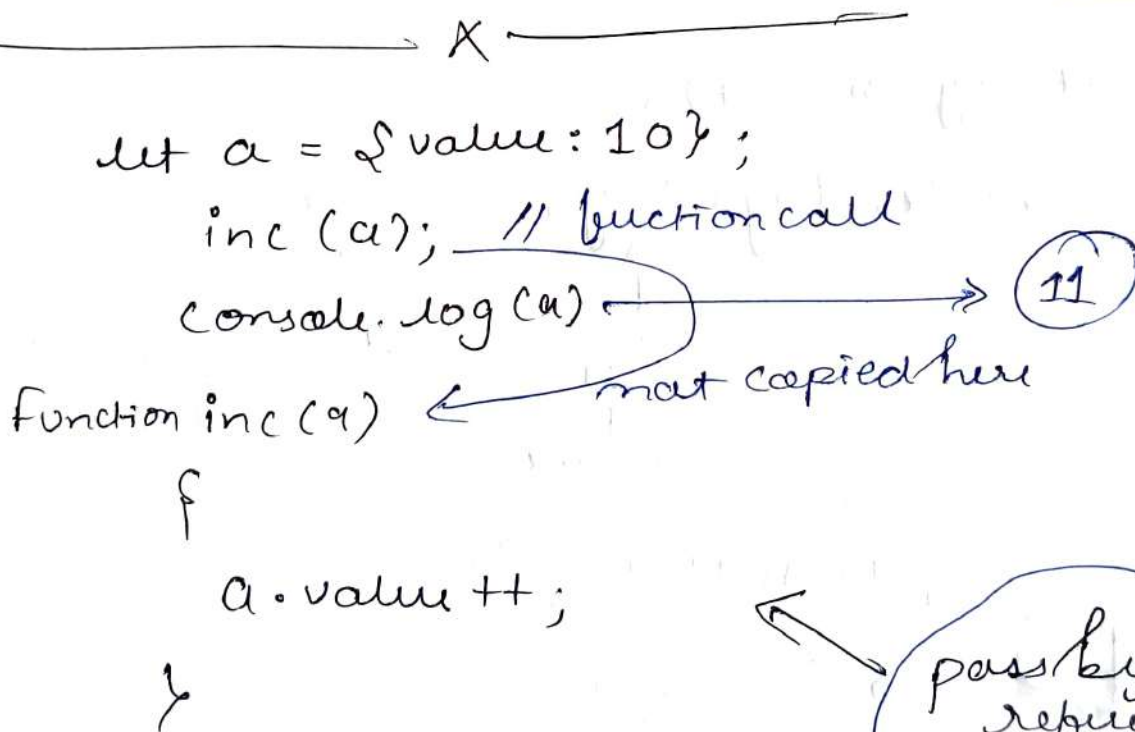
In objects, address is passed
so, both will represent the memory
location.

NOTE :- primitives are copied by their
value

References are copied by their
address.



pass by value concept
copy in primitive



pass by reference
in same address

Iterating through Objects

- 1) For-in loop
- 2) For-of loop.

For-in loop

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

```
for (let key in rect) {
```

```
  console.log(key, rectangle[key]);
```

```
}
```

↑
to
access
key
name

↑
to access value
of key

For of → doesn't work in object
→ iterables
→ Arrays
maps.

For of in Object! - (HACK)

```
for (let key of Object.keys(rect)) {
```

```
  console.log(key);
```

```
}
```

↓
gives key name
use
Object.entries(rect)
for values too

✓ We can use if else to know the property is present or not-

```
if ('length' in rect) {  
    console.log("Yes")  
}  
else {  
    console.log("No")  
}
```

Object Cloning (Same to same one more)

+ iteration
+ Assign
+ Spread } rules.

1) iteration.

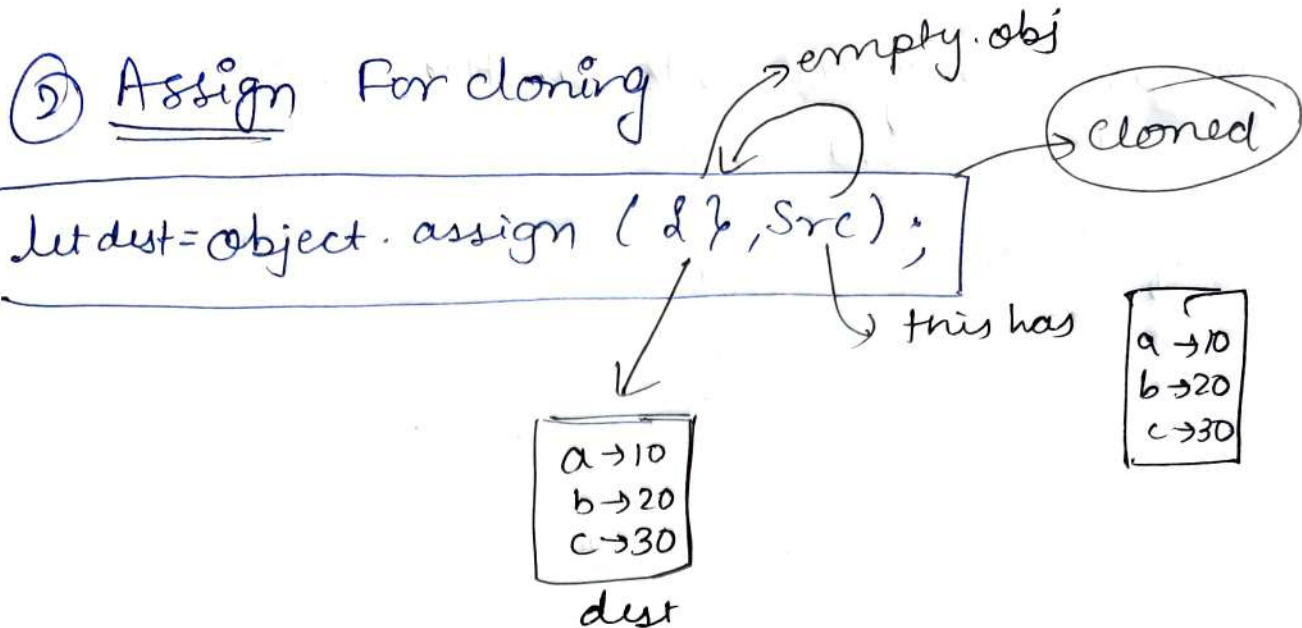
```
for (let key in Rectangle)  
{  
    console.log(key, Rectangle[key]);  
}
```

let obj2 = {};

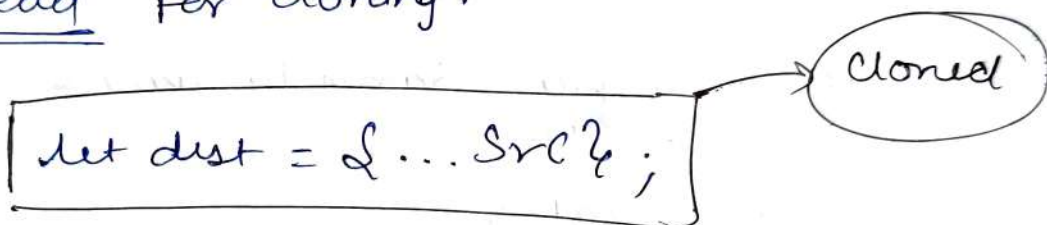
↑ we will copy all key & value of Rectangle in Obj2 one by one. At first it will be empty.

Cloned

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



⑤ Spread For cloning.



Garbage Collection :-

Find old variables/constants which are not in use and automatically deallocates their value

↓
done by
(Garbage collector)
↑
tool in JS

✓ we have no control over Garbage Collector, runs in Background itself.

JavaScript - class - 3

Basics - 3 (Inbuilt Objects & Arrays)

① Math Object :-

↳ Inbuilt Object
for mathematical functions.

- Math.random()
↳ generates Random no. b/w 0 & 1
- Math.max(2, 1, 4, 3)
↳ for maximum no.
- Math.round(1.8)
↳ round of 1.8 i.e 2
- Math.abs(-2)
↳ absolute, returns positive for positive
and positive for negative.
here, 2 will be returned.

② String Object :-

There are two types of strings in JS

↳ String → primitive.

↳ let name = 'Turwashi';

String → object

String
Constructor
Function.

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

We can convert primitive String to object using `.` notation.

`name.length`, `name.includes('Tus')`
`name.startsWith('Tus')`, `name.endsWith('ash')`;
`(name.toUpperCase())`; `name.toLowerCase()`;
`name.trim()`, `name.replace('Tus', 'Pus')`;
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 newline ② \n

But another alternative without
using \ slashes
we use

Template Literal

↳ Back Tick is used.



```
let msg = `This is  
my message`;
```

↓
Same will be the
output.

also we can add variable, in backtick using \$

```
let msg = `This  
is my message,  
Hello ${name}`;
```



In a same order

& name = 'Turvash'
will be printed.

④ Date and Time :-

Date :-

~~let date = new Date();~~

(1) let date = new Date();
 clog(date);

↑
current date & Time

(2) let date2 = new Date('June 20 1998 07:15');
 clog(date2);

(3) let date3 = new Date(1998, 6, 20, 7);

↑
year

↑
month
including
starts
from 0

↑
date

↑
Time
(hr)

also
change year,
↓

date3.setFullYear(1947)
 clog(date3);

6 is july.

Function / method (Getter / Setter)

When we use Function / method to
Set value \rightarrow Setter
Retrieve value \rightarrow getter

Arrays :- (Object / Reference type
collection of all types
of items)

- Adding new Elements
- Finding Elements
- Removing Elements
- Splitting Elements
- Combining Elements.

1) Creation :-

| | | | | | | |
|------------|--------------|--------------|--------------|--------------|--------------|-----------------------|
| | 0 | 1 | 2 | 3 | 4 | \rightarrow Indexes |
| | \downarrow | \downarrow | \downarrow | \downarrow | \downarrow | |
| number | | | | | | |
| let arry = | [1, | 3, | 5, | 7, | 'Tunwash'] | |

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

Insert :-

+ End
+ Beginning
+ middle

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

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

(ii) Beginning \rightarrow [8, 1, 4, 5, 7]
 \rightarrow number.unshift(8)

(iii) Middle \rightarrow
 \rightarrow number.splice(2, 0, 'a', 'b', 'c')

index deletion adding

3) Find out Number (Searching Element)

`numbers.indexOf(2);`

→ if we want to check if a number exist in an array.

`if (numbers.indexOf(10) != -1)`

`console.log('present')`

↖ Not Right way.

Good practice

`console.log(numbers.includes(7));`

↘ true / False (returns)

Adv

`numbers.indexOf(4, 2);`

↑ ↑
Search Index to
4 Start

→ -1 Ans

-1 is answer when you write index which is not present

* We have done these on primitive
Now on
References

`let courses = [`

`{ no: 1, name: 'Love', },`

`{ no: 2, name: 'Babbar' }`

`]`

`log(courses);`

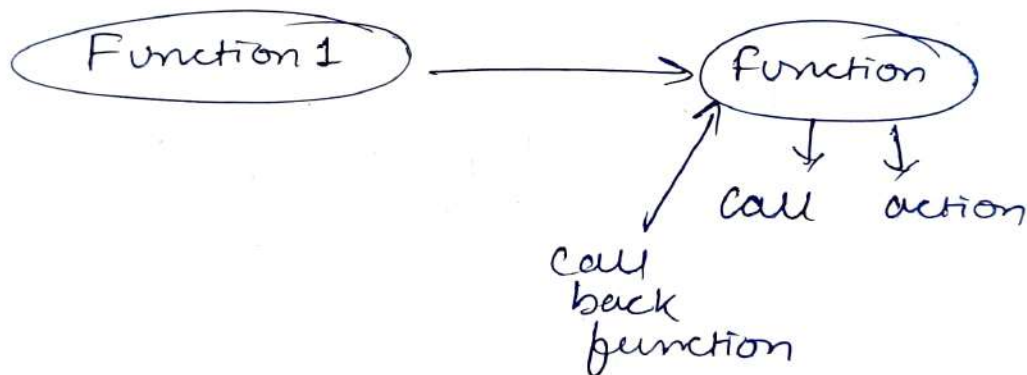
↖ array of
object is
created

In reference we cant find using
indexOf & 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 functions here,

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



```
let c = Course.find( function(course) {  
    return course.name == 'Love'  
})  
cdg(c);
```

Syntax :-

arrayName.find(

↖ predicate
function
i.e
condition to
find
object

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

Arrow Function (more concise)

✓ let course = course.find(course =>
course.name == 'Love');

we remove {
return

only when we
have 1 value
single value.

no input parameter then
arrow function
() =>

④ Removing Element :- [1, 2, 3, 4]

end → pop()

Beginning → shift()

middle → splice (3, 1)

↑
Index

↑
no.
of element
you
want to
delete

⑤ Emptying an Array :-

numbers = [1, 2, 3, 4, 5]

① numbers = []

→ empty. automatically
removed
by
garbage
collector
then,

↑
it's not
deleted still it's
stored

For deleting.

② `numbers.length = 0`

↳ this is what we do. to make array empty.

also

③ `number.splice (0 , number.length)`

↑
index

↑
no. of element
you want to
delete

④ also,

`while (numbers.length > 0)`

`numbers.pop()`

↳ using loop.

⑤ Combining & Slicing Arrays :-

`let first = [1, 2, 3];`

`let Second = [4, 5, 6];`

using `concat()` → method.

`let combined = first.concat(Second);`

Combine

slicing,
using slice() method

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

↓
slice this

slice ()
 ↑ ↑
 Start end
 Index Index

(x, y)
↓
range
where x is
included
y is
excluded.

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

0 1 2 3 4 5
 ↓ ↓
 include exclude

slice (2, 4) to get (3, 4)
 ↓
 slice

* if we give one parameter

slice (2)

↳ then from 2nd index
all removed.

* if slice ()

↳ copy of original array
called as Full slicing.

* Spread Operator

let first = [1, 2, 3]

let second = [4, 5, 6]

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

also to add

let combined = [...first, 'a', ...second, 'b']

to copy

let another = [...Combined]

* Iterating an Array =

loop

→ for of loop is on
iterables

→ foreach → also

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

for of [for (let value of arry) {
 log(value)
}

Per Each [~~to~~ arry.forEach (function (number) {
 log(number)
});

(DO change this
to ~~array~~ arrow
Function)

* Joining Arrays

num = [1, 2, 3]

to join them
like (1, 2, 3)

using join() method.

num = [1, 2, 3, 4]

const Joined = num.join()

log (Joined)

→ (1, 2, 3, 4)

Split() method
Creates an array.

let msg = "This is my message";

let parts = msg.split(" ");

log (parts)

['This', 'is', 'my', 'message']

~~let/Joined = parts. ('-')~~
~~log (Joined)~~

* Sorting Arrays

↳ using sort() method

Sort is to arrange in increasing or decreasing order
by default ascending order.

let num = [10, 50, 20, 60, 30]

num.sort()

log(num)

↳ [10, 20, 30, 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.

For +ve

let num = [1, 2, -3, -4]

let filtered = num.filter(function(value) {
return value >= 0

})

log(filtered)

↳ [1, 2]
Output.

* Mapping Arrays → map each element of array to something else.
↳ map() method
Same like ASCII

```
let numbers = [7, 8, 9, 10];
```

```
numbers.map(function (value) {  
    return 'Student_no' + value;  
})
```

↓

```
['Student_no 7', 'Student_no 8' ...  
...]
```

mapping with objects

```
let num = [1, 2, -3, -5]
```

```
let filtered = num.filter(value => value >= 0)  
console.log(filtered)
```

```
let item = filtered.map(function (num) {
```

```
    let obj = {value: num};  
    return obj;  
})
```

↓

```
console.log(item)
```

↑

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

JavaScript - class 4

Basics - 4

Functions :-

↳ a block of code that fulfills a specific task.

Syntax :-

1) function printCounting() {

console.log("Counting") }

} Fnc. body

* Why Functions?

↳ Reusability
to reduce Bulky codes
Remove / Reduce Bugs

Function Declaration :-

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.

② Function Assignment

↳ giving variable to a function.
(assigning)

```
let Stand = function walk () {  
    console.log("walking");  
}
```

To call → Stand() Not walk();

Hoisting Doesn't work here

↳ only for function declaration.

③ Anonymous

↳ Name of function not present

```
let Jump = function () {  
    console.log("walking")  
}
```

Call → Jump();

Function Assignment

↓
Named

```
let a = function name() {
```

```
}
```

↓
Anonymous

```
let a = function () {
```

```
}
```

Dynamic Function :-

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

1) `clg (sum (1));` // will give NaN (undefined for b)

2) `clg (sum ());` // will give NaN (undefined for a & b)

3) `clg (sum (1, 2, 3, 4, 5))` // only 1 & 2 will be taken rest will be waste

ans - (3)

Stored in Argument Object in JS.

Special Object \rightarrow 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)
```

```
console.log (ans)
```

\swarrow we can
increase this
to get new
value.

Rest Operator :- ...

\rightarrow we can handle multiple
parameters in function using
Rest operator.

This will create Array.

```
function sum (...args) {
```

```
  console.log (...args)
```

```
}
```

```
sum(1, 2, 3, 4, 5, 6);
```

\rightarrow [1, 2, 3, 4, 5, 6]
will be stored in array.

2) Function `sum(num, value, ...args) {`
`calc(...args);`

`sum(1, 2, 3, 4, 5, 6);`

Annotations for `sum(1, 2, 3, 4, 5, 6);`:

- `1` is stored in `num`.
- `2` is stored in `value`.
- `3, 4, 5, 6` are rest operator stored in `arguments(...args)`.

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

Reason: It is a last parameter after this NO parameter is allowed.

Default parameters :-

function `interest(p, r=10, y=2) {`

Annotations for `interest(p, r=10, y=2)`:

- `r=10` and `y=2` are default values.
- all rest have to be default.

`return p * r * y / 100;`

}

`calc(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 : 'Tiwari',
  lname : 'Chakraborty',
};
```

```
function fullname() {
  return ` ${person.fname} ${person.lname}`;
}
```

call (fullname());
 ↗ This is only read only function.

to manipulate

Getter, Setter :-

```
let person = {
  fname : 'Love',
  lname : 'Bebber',
```

```
get fullName() {
  return ` ${person.fname} ${person.lname}`;
}
```

```
set fullName(value) {
  let parts = value.split(' ');
  this.fname = parts[0];
  this.lname = parts[1];
}
```

```
};
```

To call:-

```
call (person.fullName);
      ^ getter
person.fullName = 'Rahul Kumar';
call (person.fullName); ← setter.
```

ERROR HANDLING

using Try & Catch block.

```
Try {  
    code, if error goes to catch  
}
```

```
catch (e) {
```

```
    // custom error message.
```

```
}
```

```
let person 1 = {
```

```
    fname : 'Love';
```

```
    lname : 'Babbar';
```

```
    get fullName() {
```

```
        return ` ${person1.fullname}  
                ${person1.lname}`;
```

```
    },
```

```
    • Set fullName (value) {
```

```
        if (typeof value !== String) {
```

```
            throw new Error("Not a string");
```

```
            let parts = value.split(' ');
```

```
            this.fname = parts[0];
```

```
            this.lname = parts[1]; } }
```

error handling {

```
try {
```

```
    person1.fullName  
    = true;
```

```
}
```

```
{ catch (e) {
```

```
    alert(e);
```

```
}
```

Scope:-

↳ lifetime or lifespan of variable is scope

```
{  
    let a = 5;  
}  
cng(a); → error
```

} block scope

// Sorting:-

let a = [10, 30, 50, 60, 20, 80]

a. Sort (function (a, b) {

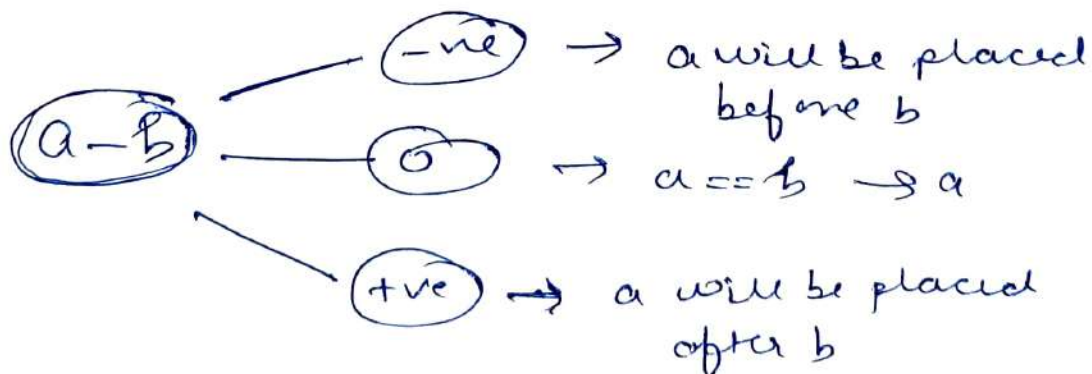
return a - b;

});

cng(a);

For ascending

b - a For descending



// Reducing 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 totalSum = arr.reduce((accumulator,  
currentValue) => accumulator +  
currentValue, 0);
```

↑
accumulator
initialized to 0

```
console.log(totalSum); → ⑩
```

working:- [1, 2, 3, 4]

accumulator = 0

current value = 1

accumulator = 0 + 1
= 1

current = 2

accumulator = 1 + 2
= 3

current = 3

accumulator = 3 + 3

current = 4

accumulator = 3 + 4
= ⑩