Turwash's Notes

7th February 2023

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) Firsty 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

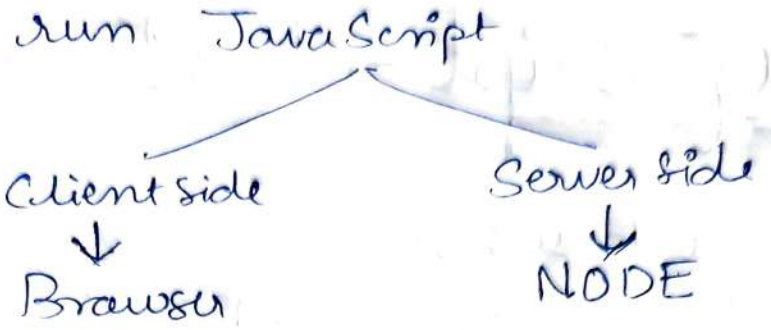
Client Side Scripting language executes on web browser.

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)



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>
    console.log("Namaste Duniya");
</Script>
```
→ to print or dock.

↑ used for client side Scripting

✓ 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 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 = "Tuwash"; (String)

let status = true; (boolean)

let b = 12.5 (Float)

↑ Variable name

Let keyword

Var keyword

var a = 12,

var name = "Tuwash"

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   (updation not allowed)
```

No redeclaration

Variable Naming

Rules

- cannot be a reserved keyword (let if X)
- meaningful
- cannot start with number (1b X)
- 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 ~~datatypes~~ items (js))
(3) Functions

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

✓ let person = {
    firstName = 'Tirwash',
    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 → love
1 → rahul
2 → sangram

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

(power)

(1) Arithmetic (+, −, *, /, %, **)
(2) Assignment (=, +=, −=, *=, /=, %=)
(3) Comparison (>, <, >=, <=, ===, !==)
(4) Ternary (condition) (condn ? val 1 : val 2)
(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
console.log(a++)
6

x++ → post increment Operator

First use the value
Second increment the value

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)

`==` v/s `===`

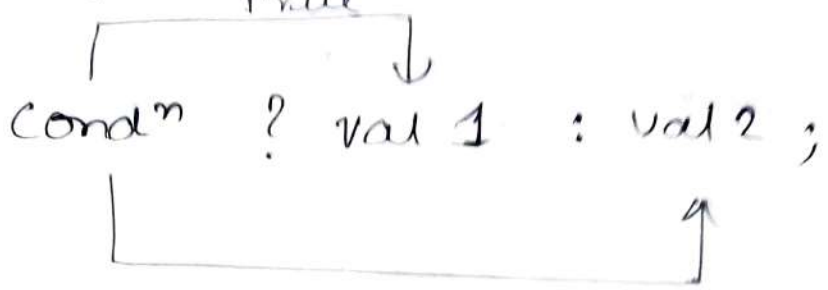
`==` → loose equality, value is same or not

```
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,

(falsy || '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 → 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

|

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 precedency.
let c = (a + ((b * d) / c))]

Control Statements :-

1) If - else
2) Switch

} two ways.

(1) if - else :- (if - elseif - else)

Single { if (condition) {
— —
}

Can be multiple { else if (condition) {
— —
}

Single { else {
— —
}

(2) Switch - case :-

Logic { input 1 → A
2 → B
3 → C

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 — updation

0 1 2 3 4

2) While loop

```
let i = 0;        <- initialization
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.

Turuvashi's Notes

9th February 2023

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

- operater to access properties of object

ex:

```
let rectangle = {
    length : 2,
    breadth : 4,
    draw : function () {
        console.log("function Draw")
    }
}
```

properties ← length, breadth

function ← draw : function ()

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 objectname.

}

OR

2) function createRectangle () {

return rectangle = {

- - -

}

write return here

Fixed values

Calling Factory function :-

```
let name = createRectangle();
```

name → Object Stored here

createRectangle() → returns object

```
Console.log(name);
```

↑ to show the function which prints object.

Input parameters for Function.

parameter

```
function createRectangle (length, breadth) {
    return rectangle = {
        length,
        breadth,
        draw() {
            alg("draw");
        }
    }
}

let rectangleObj1 = createRectangle(5, 4)
```

length

breadth.

value can change here

(2) Constructor Function

↳ we fallow pascal Notation. ✓

Camel → numberOfStudents

✓ pascal → FirstWordOfWordAllwaysCapital

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

(this → current object)

* **Constructer function**
  - defines the properties of methods
  - does not returns

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 = newRectangle(5,6)
```

a.draw()

↓ object ↓ method

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

this is 'a' 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.

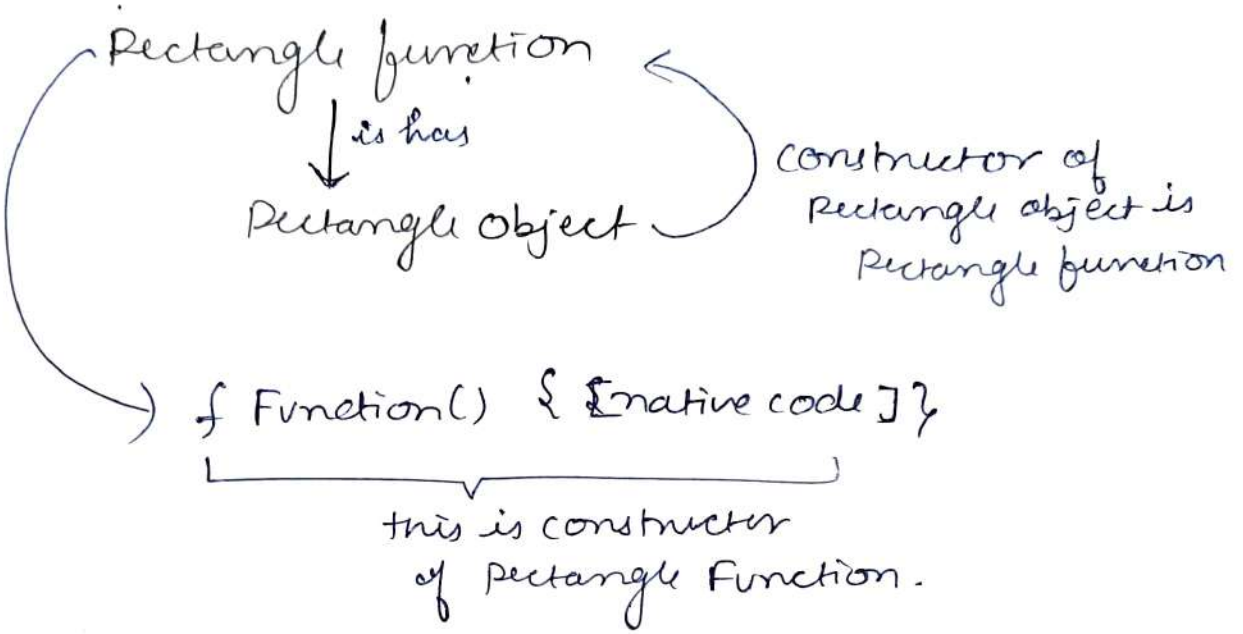
To remove

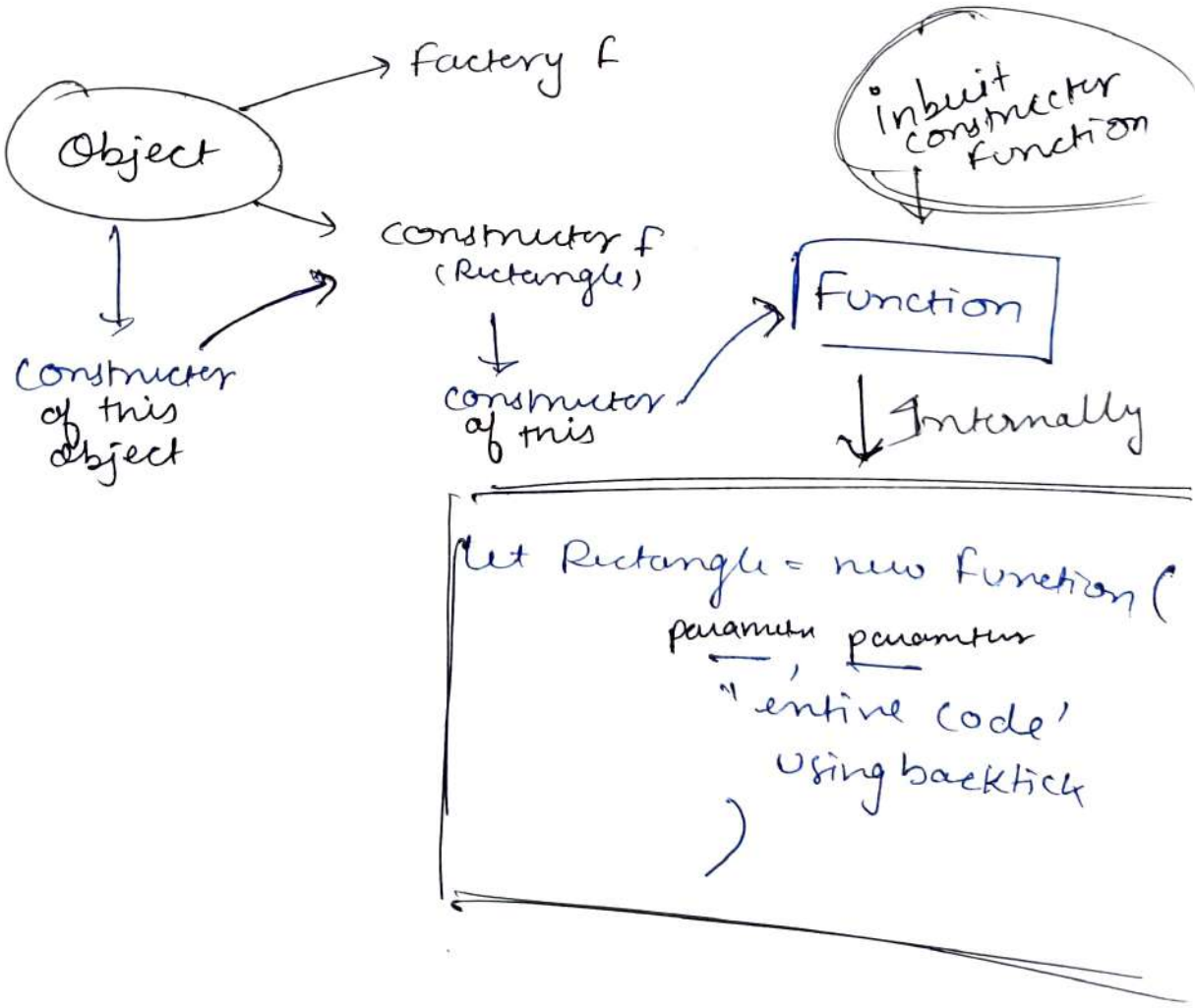
```
delete rectangleObj.color;
```

CONSTRUCTOR

Function is also an object

all object has a constructer





✓ Functions are Objects

as it have properties & entity ✓

Difference primitive type & Peference type:

primitive

```
let a = 10  → a [10]
let b = a   → b [10]

a++;
```

Here copy is created ✓

→ print(a) → 11
→ print(b) → 10

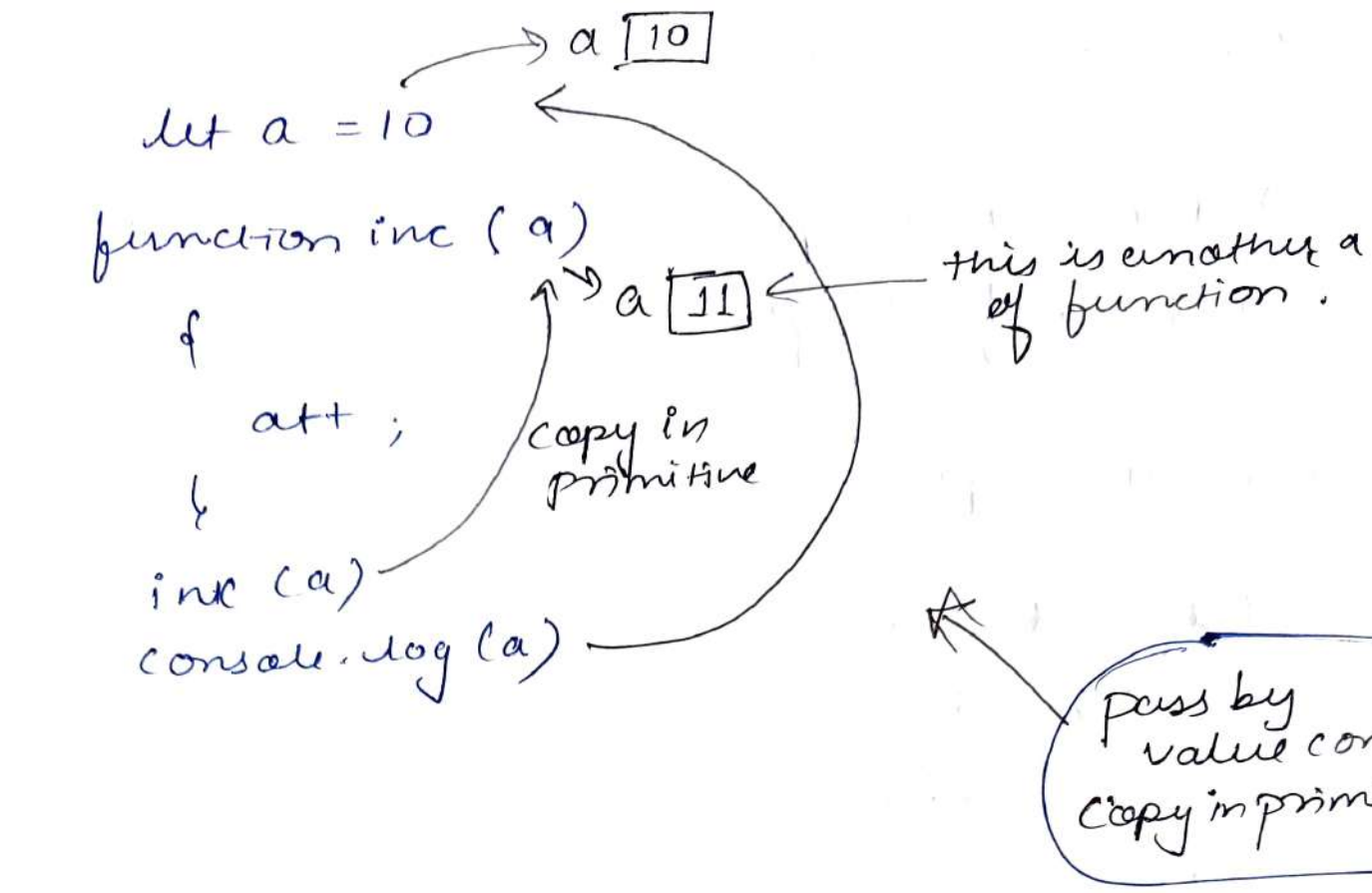
Reference

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

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.



X

```
let a = {value: 10};
  inc(a); // fuctioncall
  console.log(a)
function inc(a)
  {
    a.value++;
  }
```

11

not copied here

pass by refrence 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])
}
```

key → to access key name

rectangle [key] → 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)
}
```

object.keys(rect) → 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.

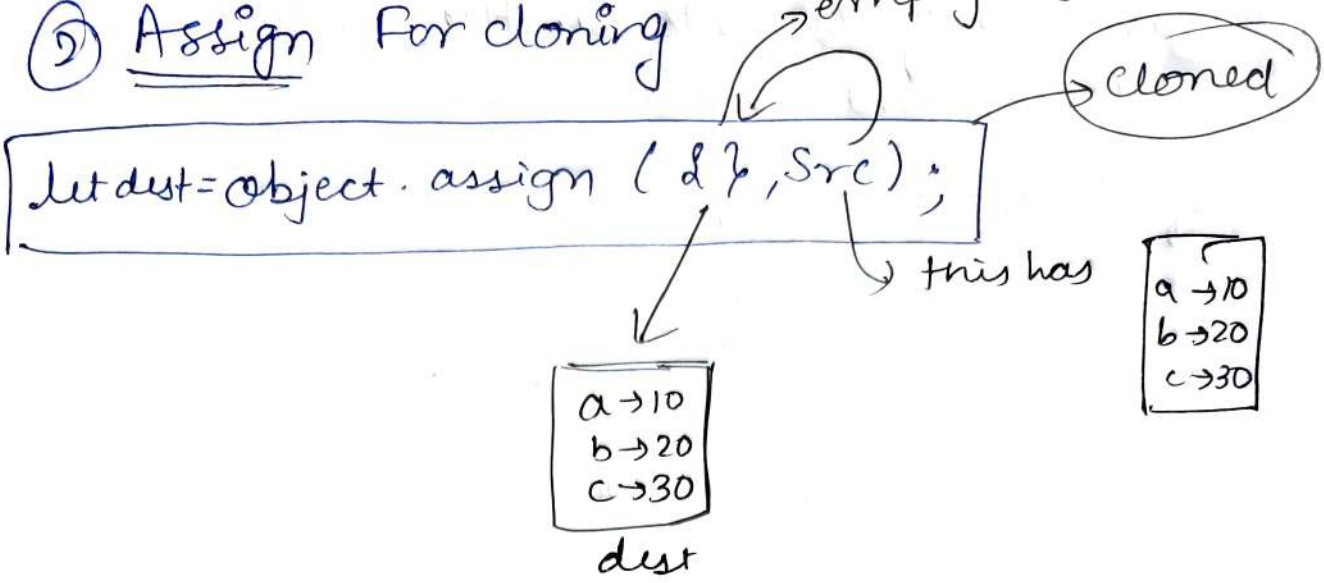
```
for (let key in Rectangle)
{
  console.log (key, Rectangle[key]);
}

let obj2 = { };
```

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

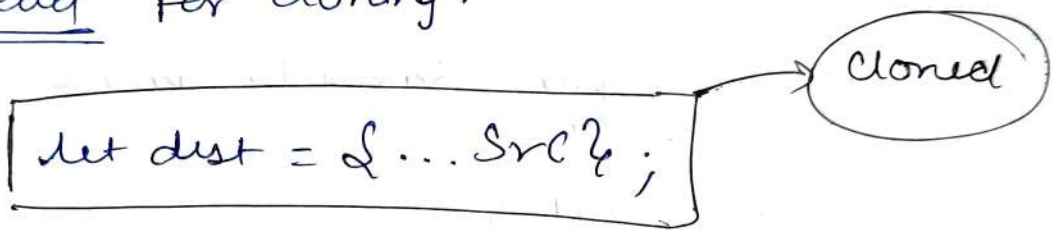
Cloned

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

② Assign For cloning



③ Spread For cloning.



Garbage Collection :-

↳ Find all 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.

Tuwashi Notes

11th February 2023

JavaScript - class - 3

Basics - 3 (In built Objects & Arrays)

(1) Math Object :-

↳ In built 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 = 'Tuwash';

String Constructor Function.

String → object
↳ 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.toUppuCase()); 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.

(\') — slash is used
these are the notations

like
for newline (\n)

But another alternative without using \ slashes we use

Template literal

by BackTick 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 messege.
Hello ${name}`;
```

↑ In same order & name = 'Turuvash' will be printed.

(4) Date and Time :-

Date :-

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

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

↖ current date & Time

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

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

↑ year ↑ month indexing Starts from 0 6 is july. ↑ date ↑ Time (hr)

also change year,

↓

```
date3.setFullYear(1947)
clg(date3);
```

Function / method (Getter / Setter)

When we use Function / method to

Set value ⟶ Setter

recieve value ⟶ 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   → Indexes
                ↓   ↓   ↓   ↓        ↓
let numbers = [ 1,  3,  5,  7,  'Tuwash' ]
```

2) Adding / pushing new Element :-

we can access array using Indexes

numbers[0] → 1 (value in index 0)

Insert :-

- End
- Beginning
- middle

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

(i) End → [1, 4, 5, 7, 9]

→ numbers.push(9)

(ii) Beginning → [8, 1, 4, 5, 7]

→ numbers.unshift(8)

(iii) Middle

→ number.splice(2, 0, 'a', 'b', 'c')

index, deletion, adding

3) Find out Number (Searching Element)

```
number.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)

Also

```
numbers.indexOf(4, 2);
```

↑ Search 4 ↑ Index to start

→ (-1) Any

(-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' }
]

clg(courses);
```

↖ Array of object is created

In reference we cant find using indexOf & includes because Searching in reference is not same as primitive.

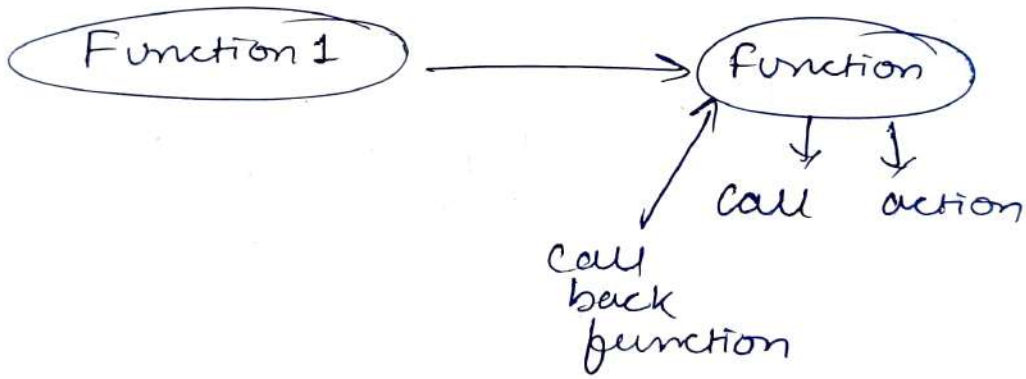
For primitive it search by value

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'
})
clg (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)
  - 3 ↑ index
  - 1 ↑ no. of element you want to delete

⑤ Emptying an Array :-

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

① numbers = [ ] → empty.

↑ its not deleted still its stored

→ automatically removed by garbage collector then,

For deleting:

(2) numbers.length = 0

↳ this is what we do to make array empty.

also

(3) numbers.splice (0 , number.length)

↑ index

↑ no. of element you want to delete

(4) also,

```
while (numbers.length > 0)
    numbers.pop()
```

→ using loop.

(6) Combining & Slicing Arrays :-

Combine

```
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]

↓

Slice this

Slice (__ __) → (x, y)

↓ range where x is included y is excluded.

↑ Start Index

↑ end Index

[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) {
    clg (value)
}
```

For Each

```
arry.forEach (function (number) {
    clg (number)
});
```

(DO change this to 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()

clg (joined)

↳ 1,2,3,4

Split() method

Creates an array.

let msg = "This is my message";

let parts = msg.split(" ")

clg (parts)

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

let joined = parts.('-')

clg (joined)

* Sorting Arrays

by using sort() method

Sort is to arrange in increasing or decreasing order

by defaut ascending order.

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

num.sort()

clg (num)

↳ [10, 20, 30, 50, 60]

also reverse using

num.reverse()

[60, 50, 30, 20, 10]

→ We cant 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
})

clg (filtered)

↳ [1, 2]

output.

* Mapping Arrays → map each element of array to something else.

↳ map() method

Same like ASCII

a → 97

b → 98

Same →

```
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)
clg(filtered)
let item = filtered.map(function(num){
    let obj = {value: num};
    return obj;
})
clg(item)
```

↑

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

Turwash Notes

12th February 2023

JavaScript - Class 4

Basics - 4

Functions :-

↳ a block of code that fulfills a specific task.

Syntax :-

keyword → function ← Functionname

```
1) function printCounting () {
       console.log ("counting")
   }
```

Fnc. body

* Why Functions ?

↳ Reusability
to reduce Bulky codes
Remove / Reduce Bugs

Function Declaration :-

I/P parameter

(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();

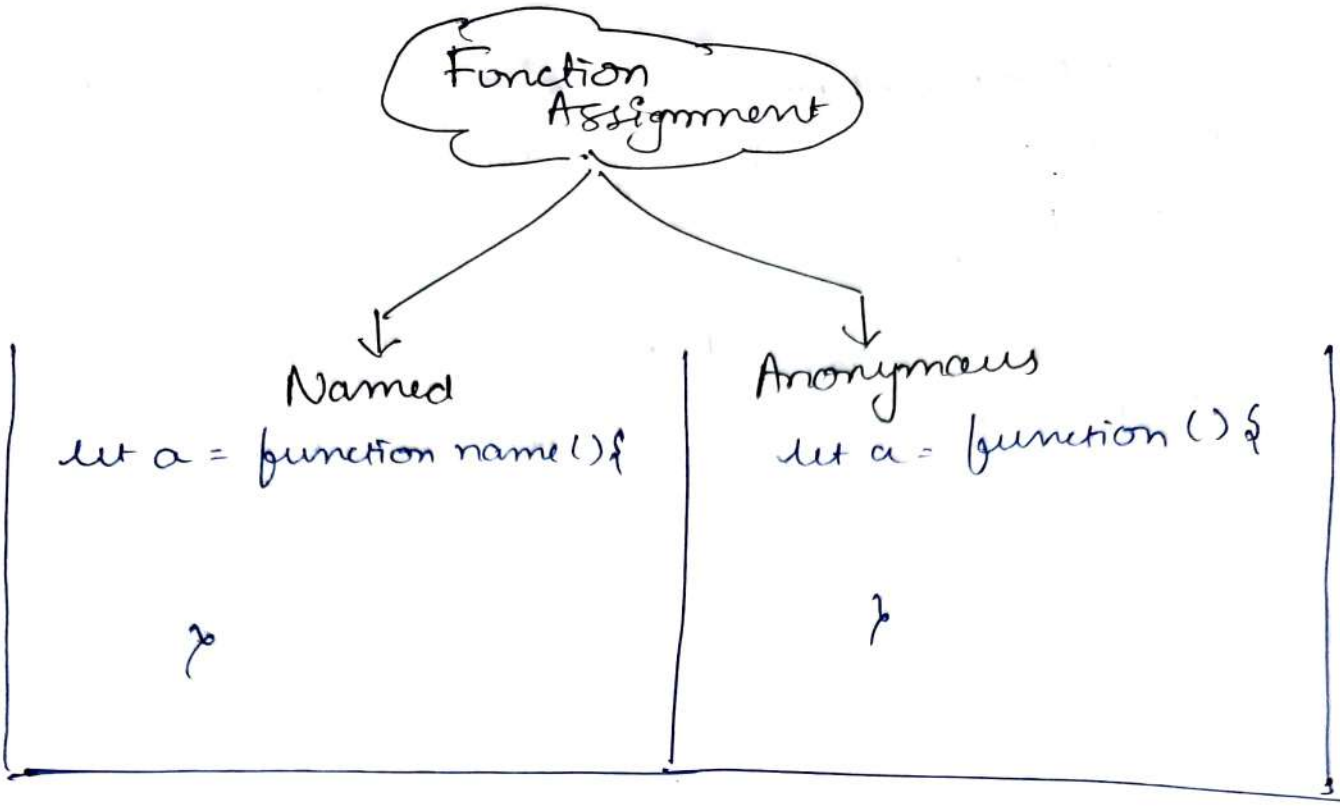
Hoisting Doesn't works here

↳ only for function declaration.

③ Anonymous

↳ Name of function not present

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

call → Jump();
```



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 (undefind 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 → 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)
```

← we can increase this to get new value.

Rest Operator :- ...

↳ we can handle multiple parameters in function using Rest operator.
This will create Array.

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

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

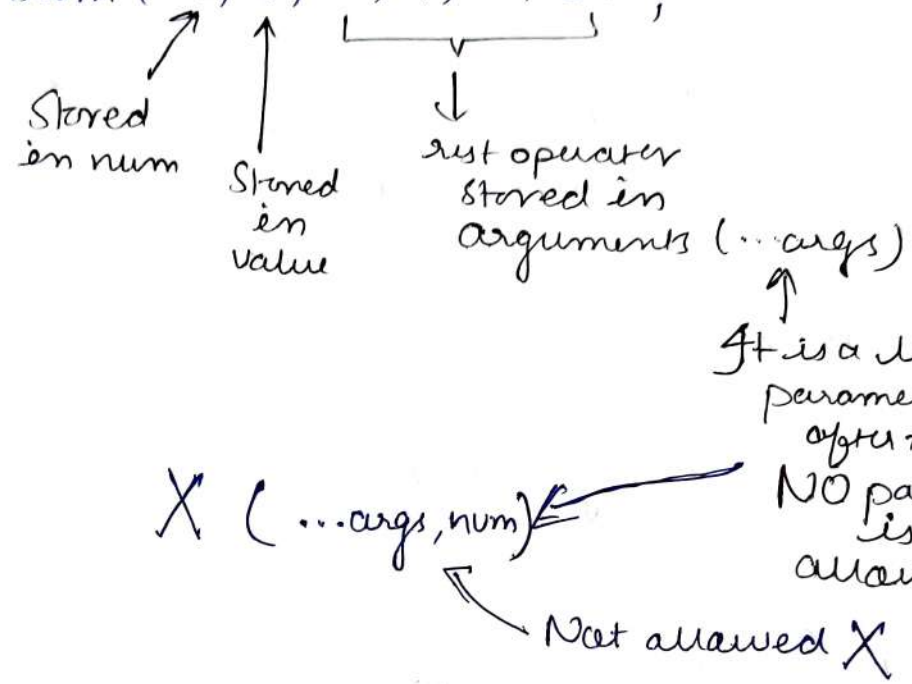
↳ [1, 2, 3, 4, 5, 6.]
will be stored in array.

2) function sum(num, value, ...args) {

clg(...args);

}

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



Default parameters :-

default default

function intrest (p, r=10, y=2) {

all rest have to be default

return p*r*y/100;

}

clg (intrest (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 : 'Tuwash',
    lname : 'Chakraborty',
};

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

clg (fullname());
```

↖ This is only read only function.

to manipulate.

Getter, Setter :-

```
let person = {
    fname : 'Love',
    lname : 'Babbar',
    get fullName() {
        return `${person.fname} ${person.lName}`
    },
    set fullName(value) {
        let parts = value.Split(' ');
        this.fName = parts[0];
        this.lName = parts[1];
    }
};
```

To call :-

```
clg (person.fullName);
        ↖ getter
person.fullname = 'Rahul Kumar');
clg (person.fullName); ← Sitter.
```

ERROR HANDLING

using Try & Catch block.

```
Try {
    code, if error goes to catch
}
catch (e) {
    // custom error message.
}
```

```
let person1 = {
    fname : 'Love',
    lname : 'Babbar',
    get fullName() {
        return `${person1.fname}
                ${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;
}
console.log(a); → error
```

block scope

// Sorting:-

```
let a = [10, 30, 50, 60, 20, 80]
a.sort(function(a, b) {
    return a - b;
});
console.log(a);
```

a - b → For ascending

b - a → For descending.

a - b
- -ve → a will be placed before b
- 0 → a == b → a
- +ve → a will be placed after b

// 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 parameter

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
currentValue = 1
accumulator = 0 + 1
= 1
current = 2
accumulator = 1 + 2
= 3
current = 3
accumulator = 3 + 3

current = 4
accumulator = 6 + 4
= (10)